Trusted Accounting in Peer-to-Peer Environments – A Novel Token-based Accounting Scheme for Autonomous Distributed Systems

Vertrauenswürdiges Accounting in Peer-to-Peer Umgebungen – Ein Neuartiges, Token-basiertes Accounting-Schema für Autonome Verteilte Systeme

Zur Erlangung des Grades eines Doktor-Ingenieurs (Dr.-Ing.)

genehmigte Dissertation von Dipl.-Wirtsch.-Ing. Nicolas Christopher Liebau, geboren am 24.10.1973 in Fulda

September 2008 — Darmstadt — D 17

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Elektrotechnik und Informationstechnik
Fachgebiet Multimedia Kommunikation

Trusted Accounting in Peer-to-Peer Environments – A Novel Token-based Accounting Scheme for Autonomous Distributed Systems
Vertrauenswürdiges Accounting in Peer-to-Peer Umgebungen – Ein Neuartiges, Token-basiertes Accounting-Schema für Autonome Verteilte Systeme

genehmigte Dissertation von Dipl.-Wirtsch.-Ing. Nicolas Christopher Liebau, geboren am 24.10.1973 in Fulda

1. Gutachten: Prof. Dr.-Ing. Ralf Steinmetz
2. Gutachten: Prof. Dr.rer.nat. Burkhard Stiller
3. Gutachten: Dr. Andreas Mauthe

Tag der Einreichung: 24.09.2008
Tag der Prüfung: 21.11.2008

Darmstadt — D 17

Abstract

Communication systems based on the peer-to-peer (p2p) paradigm present what is likely the most important development in Internet technology in recent years. In spite of the position of p2p systems as the largest source of traffic on the Internet, their commercial success is still limited. The basic tenets of p2p systems are cooperation among peers and completely decentralised communication. However, these can result in nontransparent actions, as well as opportunistic behaviour of the individual peers.

Thus, the implementation of commercial p2p applications requires basic mechanisms to record transactions in the systems, i.e. the resource and service consumption, which will be used for charging, incentives, and control. These basic functionalities can be achieved by an accounting system that ideally should be fully distributed in keeping with the p2p spirit of such a system. Furthermore, it needs to be trustworthy so that the system cannot be misused by individual peers or groups of peers for gaining an undue advantage. Therefore, linking distributed *accounting* with distributed *trustworthiness* and distributed *collaboration control* presents a crucial challenge for the advancement of p2p systems.

This dissertation researches this challenge by *demonstrating the feasibility of fully distributed, trusted accounting in p2p systems with intrinsic automatic cooperation control by presenting and evaluating the token-based accounting scheme.*

The token-based accounting scheme's framework introduces tokens as a combination of permission objects and receipts. Permission objects grant to peers the right to consume services and resources. When a peer consumes them, it "spends" tokens, and tokens becomes receipts which contain accounting information about the transaction. This process implements effective accounting with intrinsic cooperation control.

The token-based accounting scheme's system architecture is composed of four building blocks: *Token structure*, *transaction protocols*, *token aggregation*, and *detection of double spending*.

The token structure ensures the authentication and integrity of accounting information, as well as the non-repudiation of transactions.

The trustworthy transaction protocol introduces a *novel transaction procedure* that removes the benefits of defrauding the transaction partner but does not require the use of a third trusted party.

Token aggregation swaps foreign tokens for new own tokens using a truly decentralised trustworthy process. With this process, a quorum of peers establishes a *novel distributed basis of trust for p2p systems*. This is achieved by applying threshold cryptography in combination with proactive secret sharing and with novel mechanisms that ensure the random selection of the quorum peers. The quorum size affects the scheme's trustworthiness and is determined using a stochastic model.

Efficient detection of double spending is enabled by introducing aggregation accounts that store issuing and usage information about tokens. Aggregation accounts are located at third party peers, called account holder sets. Maintenance operations performed on the account holder sets prevent loss of data and ensure consistency of aggregation accounts. Aggregation accounts are protected against attacks and fraud attempts by concealing their location using a *novel overlay routing mechanism*. By using simulations for several churn scenarios, the required account holder set size is determined. These simulations prove the token-based accounting scheme's efficiency and the robustness of the storage mechanisms.

The token-based accounting scheme was simulated in detail by varying the relevant parameters, i.e., quorum size, account holder set size, churn, and system size. The simulation results demonstrate the viability and efficiency of the novel token-based accounting scheme. Its applicability is shown in two application scenarios.

Kurzfassung (Deutsch)

Kommunikationssysteme basierend auf dem Peer-to-Peer-Paradigma stellen die wohl bedeutsamste Weiterentwicklung der Internettechnologie in den vergangenen Jahren dar. Obgleich sie heute für den Großteil des Datenaufkommens im Internet verantwortlich sind, ist ihre kommerzielle Nutzung noch gering. P2P-Systeme basieren auf der Grundidee der Kooperation der teilnehmenden Peers und der vollständig dezentralen Kommunikation. Allerdings bleiben dadurch Handlungen und Verhalten der einzelnen Peers intransparent.

Zur Realisierung verlässlicher, kommerziell nutzbarer P2P-Applikationen werden daher grundlegende Mechanismen benötigt, welche die Transaktionen im System, d.h. Ressourcen- und Dienstnutzung, erfassen, und somit als Basis für Abrechungs-, Anreiz- und Kontrollverfahren dienen.

Diese Grundfunktionalitäten können durch ein so genanntes Accounting-System ermöglicht werden, das aber vollkommen verteilt funktionieren muss, um die Vorzüge des P2P-Ansatzes aufrechtzuerhalten. Weiterhin muss ein solches System vertrauenswürdig sein, d.h. einzelne Peers oder Peer-Gruppen dürfen es nicht manipulieren können, um unerlaubte Vorteile zu erhalten. Daher ist die Kombination eines verteilten Accountings mit verteilten Mechanismen zum Erhalt der Vertrauenswürdigkeit und verteilter Kooperationskontrolle eine entscheidende Herausforderung für die Weiterentwicklung von P2P-Systemen.

Die vorliegende Dissertation erforscht diese Herausforderung, indem die technische Möglichkeit eines *vollkommen verteilten, vertrauenswürdigem Accounting in P2P-Systemen mit inhärenter Kooperationskontrolle* bewiesen wird, indem das neuartige *Token-basierte Accounting-System* vorgestellt und evaluiert wird.

Das Rahmenwerk des Token-basierten Accounting-System benutzt sogenannte Tokens als eine Kombination aus Erlaubnis und Quittung. Ein Token verleiht dem Peer das Recht, ausgewiesene Dienste und Ressourcen zu beanspruchen. Beansprucht er diese, gibt er den Token aus und der Token wird zur Quittung, der Accounting-Informationen über die Transaktion enthält. Damit wird ein effektives Accounting mit inhärenter Kooperationskontrolle umgesetzt.

Die Architektur des Token-basierten Accounting-Systems besteht aus vier Bausteinen: Token-Struktur, Transaktionsprotokolle, Token-Aggregation und das Feststellen von mehrfachem Ausgeben.

Die Token-Struktur stellt die Authentifizierung und Integrität der Accounting-Informationen sicher sowie die Nachweisbarkeit von Transaktionen.

Das vertrauenswürdige Transaktionsprotokoll benutzt einen *neu entwickelten Transaktionsprozess*, der den Gewinn aus betrügerischem Verhalten innerhalb einer Transaktion eliminiert, ohne dabei auf eine dritte Partei zurückzugreifen.

Im Token-Aggregation-Prozess werden fremde gegen neue Tokens in einem vollkommen verteilten, vertrauenswürdigen Prozess eingetauscht. Dabei bildet ein Quorum aus Peers eine *neuartige, verteilte Vertrauensbasis für P2P-Systeme*. Dies wird möglich, indem Schwellenwertkryptographie in Kombination mit proaktiven Secret Sharing-Techniken und neu erforschten Mechanismen angewendet wird, die die zufällig Auswahl des Quorums garantieren. Die effiziente Anwendung von proaktiven Secret Sharing-Techniken in P2P-Systemen wird mittels Simulationen belegt. Die Quorumgröße beeinflusst die Vertrauenswürdigkeit des Accounting-Systems; sie wird mit Hilfe eines stochastischen Modells bestimmt.

Die Einführung von Aggregationskonten ermöglicht die effiziente Feststellung des mehrfachen Ausgebens von Tokens. Aggregationskonten speichern Informationen über Ausstellung und Nutzung von Tokens und befinden sich auf dritten Peers, der sogenannten Kontohaltergruppe. Protokolle zur Instandhaltung der Kontohaltergruppen vermeiden Datenverlust und stellen die Konsistenz der Aggregationskonten sicher. Die Aggregationskonten sind gegen Angriffe und Betrugsversuche durch ein *neues Overlay-Routing-Verfahren* gesichert, der ihre *genaue Lage im P2P-System verbirgt*. Die benötigte Größe

der Kontohaltergruppe wird mittels Simulationen mit verschiedenen Szenarien für das Hinzukommen und Wegfallen von Peers bestimmt. Die Simulation des Token-basierten Accounting-Systems beweißt die Effizienz und Robustheit des Speichermechanismus.

Das Token-basierte Accounting-System wurde im Detail simuliert, wobei die relevanten Parameter (Quorumgröße, Kontohaltergruppengröße, Dynamik des Hinzukommens und Wegfallens von Peers, Systemgröße) variiert wurden. Die Simulationen belegen die Gültigkeit und Effizienz des neuartigen Token-basierten Accounting-Systems für P2P-Systeme. Seine Anwendbarkeit wird in zwei Szenarien gezeigt.

Acknowledgments

First and foremost, I would like to thank my supervisor Prof. Ralf Steinmetz for his support, advice, and for running the Multimedia Communications Lab which proved to be a valuable and productive work environment. Also, I want to thank my co-supervisors Prof. Burkhard Stiller and Dr. Andreas Mauthe for their great help and advice over the years, and for always responding immediately to my questions. The same is true for Dr. Oliver Heckmann. Thank you!

Many thanks also go to my colleagues from the MMAPPS project, in which I learnt so much and got the inspiration for this work. I want to give my special thanks to Ben, Burkhard (again), Costas, David, Huw, Jan, and Vasilios.

While I finished my thesis I had awesome support from my colleagues Sandra, Kalman, Sebastian, Konstantin, and Osama; special thanks to Sandra for taking over so much work! Thank you to my students who helped my with this work, especially Andreas, Ramon, and Oliver. And thank you Karola, Moni, Sabine, Frau Ehlhardt, Frau Kolb, Frau Scholz-Schmidt – you add a lot of life to the lab! Matthias, Kalman, and Matthias – thanks for the help during the last night!!!

I am especially grateful to my friends for their mental (going out) and physical (sport) support. Especially: Oli, thank you for everything! Andreas, too bad you moved to Lancaster; Allison, thanks you for being a real friend, for the proof reading, and, of course, for sending me cookies; Tom, it is great that you live so close to my place. Also, I want to thank my mother and brothers for being there.

Finally, a big thank you to all developers of peer-to-peer applications! Your software helped me a lot finding so nice things for distraction ;-) And thank you to the Skype developers for making it so easy to stay in contact with my long-distance friends :-)

Contents

1 Introduction

In 1999 the Internet started to experience a revolution: Napster (Wik08c) was the first *peer-to-peer (p2p) application* that allowed end users to directly exchange files. With it, it was no longer necessary for users to first upload their files onto a server before others could download them. This simple illustration explains the fundamental difference between the two dominating communication and systems paradigms in the Internet today: client/server and peer-to-peer.

In traditional client/server systems, a client requests services from a server. The server hosts these services, provides the resources for hosting and delivery, and controls the delivery process. Client/server systems have been developed since the beginning of the Internet. In order to illustrate the characteristics of client/server systems consider the following examples: The server farms of Google and Yahoo demonstrate that client/server systems are highly scalable. Online banking demonstrates that client/server systems can be highly secure. Content Delivery Networks demonstrate that a distributed network of servers can be used to distribute large amounts of data to end users.

However, about 60% to 80% of the traffic in the Internet today originates from peer-to-peer applications (Has05). p2p systems involve direct communication and sharing of resources amongst end-user computers called "peers". The term "peer" stems from two main characteristics: viz. the equality of peers in the sense that they provide services to other peers as well as consume services from other peers. Peers are also autonomous. Therefore, as a sub-class of distributed system, p2p systems belong to the class of autonomous distributed systems. Ideally, p2p systems are completely decentralised and organisation follows self-organisation capabilities.

Today, the predominant amount of traffic in the Internet stems from p2p file sharing applications such as Gnutella (Cli08), KaZaA (Sha04), eDonkey (Met04), and BitTorrent (Coh03), used by millions of users world wide to share digital content. This demonstrates two major properties of p2p systems; they are highly scalable, and they do not rely on central resources. These also constitute the main differences between p2p and client/server when building highly scalable systems. When the load on a client/server system grows, it has to be extended by adding resources. In contrast, while a p2p system grows, the available resources also increase. Thus, due to the decentralisation p2p systems can be deployed and operated at a fraction of the cost required for client/server systems (see e.g. (LPG+07)).

However, the disadvantage of decentralisation is a loss of control and loss of determinism with regard to system state and behaviour. There is no entity in a p2p system where such information would be available. Furthermore, there is no trusted entity available within the system, which commonly is required for providing secure services in the Internet. Therefore, today p2p systems are still limited in the application field. Hence, the commercial success of p2p systems has been very limited.

The strong advantages the p2p paradigm brings about in terms of scalability, ease of deployment, and operational costs compared to client/server systems motivates research to find new mechanisms that enable p2p systems to enjoy the same overall performance as client/server systems have today.

1.1 Motivation

In p2p systems, the combination of decentralisation and strong autonomy of peers brings about a number of difficult challenges.

Due to the decentralisation, there is no information available about peers' actions in the system, e.g., which user consumed which services or resources, and which user(s) provided them. Therefore, key information required to charge customers for commercial services is missing; typically such information is called *accounting information*.

Today, secure services are designed using central, highly trusted servers. Examples are charging and billing services as used in electronic commerce. However, an ideal p2p system does not have any central trusted component, since this would encumber the systems unlimited scalability. Hence, an accounting mechanism for p2p systems also has to resolve the apparent contradiction of decentralisation and *trust*.

Another challenge that p2p systems face is closely related to the lack of information. p2p applications rely on cooperation; the peers provide the resources required to operate the p2p system, and also provide services to each other. However, due to the lack of information about resource and service provisioning, people using p2p systems feel anonymous and unobserved, which leads to opportunistic behaviour, the so called free-riding phenomenon (AH00). Peers try to maximise their own profit by providing as little resources and services as possible to others. This obviously results in a reduced overall system performance. Hence, high performing p2p systems require a mechanism for *cooperation control*.

Thus, the key motivating question for this dissertation is: Is it feasible to overcome these challenges and *design a fully decentralised, trusted accounting system with intrinsic decentralised cooperation control* while preserving the technical advantages of p2p at the same time?

1.2 Trust, Accounting, and Cooperation Control

The key challenges of this dissertation are to design a fully decentralised accounting scheme that is trusted without relying on a trusted entity, and also has an inherent mechanism for cooperation control. In novel p2p application scenarios such an accounting scheme provides trusted accounting information that is used for charging and billing for commercial services, as basis for an efficient cooperation control for free community services, or even as a combination of both.

To visualise the key challenges, trust in distributed systems as well as two different abstract application scenarios are discussed.

The focus of the first application scenario is commercial p2p applications, for example, a commercial p2p video distribution platform. As shown in (LPG+07), centralised distribution of large files like videos bear high distribution costs in terms of traffic. Here the use of peer-to-peer systems can save up to 90% of distribution traffic compared to the use of client/server systems.

The focus of the second application scenario is community applications where users collaborate in order to achieve a common benefit, such as a distributed version of the Wikipedia encyclopedia. Due to the central hosting in data centres, such community applications rely on donations. This would be superfluous with a change to p2p.

1.2.1 Trust, Accounting, and Cooperation Control in P2P Application Scenarios

In order to provide highly trusted services, such as charging, billing, or online banking, the state-of-the-art in distributed system relies on central trusted entities. Obviously, trust is related to security, however, both are defined differently. Typically, in information systems the term "security" refers to information security. Information security comprises the three protection objectives authentication, integrity, and non-repudiation (Sch96), which are also relevant for an accounting mechanism. A trusted entity typically provides these protection objectives; however, this is incomplete, as the term trust comprises more. (see also Appendix C.1).

The trustworthiness of information systems comprises in addition to information security also functional security and "benevolence" (cf. (BHS02)). Functional security is the attribute of a system that the

realised as-is functionality of the system components is consistent with the specified to-be functionality. Benevolence is the extent to which the users believe that the application provider intends to do good things rather than just act out of profit interest (see also Appendix C.1). This is an important addition for p2p systems, since the good intentions of all participating peers cannot be assumed. Thus, the challenge for p2p systems is to design a system that offers benevolence, although its components (peers) might not be trustworthy. Hence, potential cheating and collusion of peers must be considered in the design. In consequence, to design trusted p2p systems the application of state-of-the-art security mechanism will probably not be sufficient.

The processes executed when conducting commercial transactions over the Internet partition into five phases (Sch04) (see Figure 1.1). Using the example of a commercial video delivery platform these phases help to understand the tasks of accounting in commercial applications.

In Phase 1 and Phase 2 the consumer finds a video he would like to watch, which is offered by the provider. In p2p systems this functionality is typically provided by the p2p overlay network.

In Phase 3, the negotiation phase, provider and customer negotiate a tariff for the video delivery, as well as the applying terms and conditions. The price of a video could depend different parameters, for example, on the download time or the file size. A contract to purchase, often in form of a Service Level Agreement (SLA), is created between service provider and customer.

In Phase 4, the video is delivered and the consumer pays the agreed price. In order to calculate the price according to the negotiated tariff, the charge must be calculated and a bill needs to be created stating the required payment. Billing functionality is often provided by the applied payment system. For calculating the charge, it is important that it is correctly recorded if the provider provided the service according to the SLA. The customer must not be able to dispute a correctly provided service provisioning. Equally, the provider must not be able to dispute a received payment. The required mechanism is a trusted accounting system. It accounts for the provided services and payments. Furthermore, in a p2p video deliver platform a video would probably be provided to the consumer by a set of peers. Hence, for cooperation control also the cooperation of peers is to be recorded. This information is important in order to achieve a high performance of the p2p system.

During Phase 5, the customer uses the acquired service or good. For example, the customer watches the video. During this phase, providers can perform customer support and customer relationship management.

Communities require participation of individual members. This is especially true for p2p applications, for example p2p file sharing communities.

In order to participate in a community, users must feel that their benefits outweigh their costs of participation. This can be reached by two alternative behaviour strategies. Either a user fully participates in the community and receives in return some reward that outweighs the costs; or a user abuses the community by using the benefits offered but providing no services in return. In (Gal02, MMA04) the so-called “Carrot and Stick” approach is presented in order to motivate users to fully participate in a community (this also means a user will act according to the rules of the community) and not abuse the privileges.

The “Stick” approach focuses on making users accountable for their actions. The “Stick” implies that users will also be punished for actions that violate the community’s norms. This requires that abuse can be detected and users can be punished. Detection of abuse can either be by technical means or by other community users. Technical detection requires accountability of user actions. Misbehaviour detected by users will be fed into a reputation system. Punishment can be any combination of psychological deterrents, like negative reputation made public, or loss of access to community services, which requires access control.

The “Carrot” approach is concerned with aspects that reward people for participation in a community, that is, conform to the community’s norms. A reward can be for example, premium access to services or an improved service quality, as well as an enhanced status within the community. Thus, for measuring

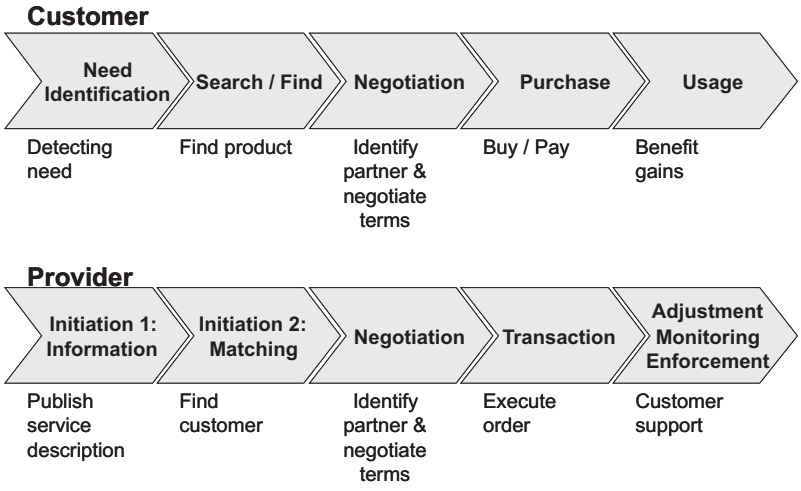


Figure 1.1: Phases of commercial transactions (Sch04)

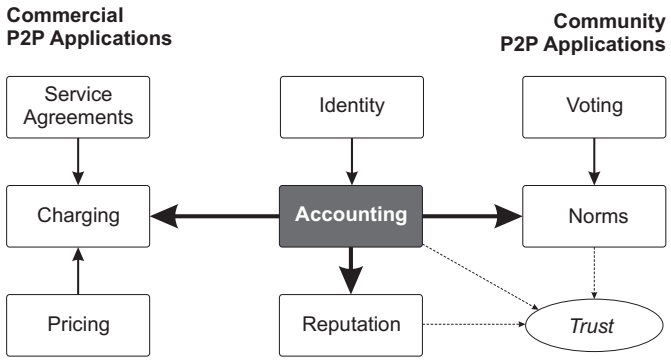


Figure 1.2: Functional requirements for commercial and community p2p applications

and expressing a persons contribution within a community, an accounting systems as well as a rating or reputation system is needed.

The technical functionality realising the “Carrot and Stick” approach can be characterised as a cooperation control mechanism.

1.2.2 Accounting as Functional Core

A strong overlap between commercial and community application scenarios on the required mechanisms for p2p systems can be observed. Accounting is a core component in both cases. Accounting provides the fundamental information other functions (such as charging, reputation, and norms) rely on. Accordingly, accounting is defined as follows:

Accounting is the process of tracing ***relevant*** *IS activities to a responsible source. Relevance is defined by the application applying accounting* (see Appendix 2.1.2).

It is crucial that accounting is trustworthy.

Figure 1.2 shows the interdependencies of application functions with accounting. Commercial applications require trusted accounting information in order to compute the charge for a service (which is required for billing and payment). Any reputation mechanism requires accounting information in order to ensure that ratings are given only when a transaction has happened, and only once per transaction. Communities require norms or rules. Compliance with rules can be detected by technical means using accounting information. Finally, accounting is central to build trust, as it delivers the basic information on the trustworthiness of other users. It is important to note that accounting requires user identification.

From this discussion, the basic requirements for an accounting scheme for p2p systems can be derived. A p2p accounting scheme should be a) fully decentralised and b) maintain a system-wide record of the information needed by other components like charging and billing about every transaction in the p2p system. In order to use accounting information for other purposes, accounting information must be efficiently and fully retrievable. c) It is especially important that an accounting scheme provides trustworthy information. As accounting is used for abuse detection as well, it should be d) combined with means for cooperation control. The preferred means of cooperation control is to constrain access to services for the users abusing the p2p system.

1.3 Challenges for Trusted Accounting and Cooperation Control in P2P Environments

Due the complexity resulting from the decentralised character of p2p systems and the autonomy of the participating peers, accounting and cooperation control are still open challenges in p2p systems.

In the context of Tanenbaum's model of the Internet (Tan03), p2p systems are placed within the application layer, just as client/server systems. They form an overlay network above the transport layer in order to interconnect the participating peers. The functionalities of trusted accounting and cooperation control lie above the overlay network and are accessed by actual applications. Trust, accounting, and cooperation control use the overlay network in order to find peers, services, and resources they require. In order to perform its functionality p2p mechanisms may not use or rely on any centralised component, e.g. a trusted server.

Specific research challenges for decentralised trust, accounting, and cooperation control mechanisms, especially compared to client/server-based solutions, are discussed following.

1.3.1 Decentralised Accounting

Both, requirements and challenges for building an accounting mechanism, differ fundamentally between client/server systems and p2p system. In client/server, a server provides the resources required for providing services and controls the provisioning of services. All relevant accounting information accumulates at the server, and is often stored in form of log-files. Accordingly, access to the accounting information is relatively simple.

In decentralised systems such as p2p, there is no central entity where accounting information is stored and can be accessed. Accordingly, accounting information must be collected in a distributed way across the complete p2p system. A peer interacts with many different peers in order to consume and provide many services. Therefore, accounting information about a single peer might also be distributed across the complete system. Accordingly, efficient access to accounting information becomes a challenge in p2p systems. For example, for billing purposes, it should be possible to access information about all relevant interactions of a specific peer that have taken place across the whole p2p system.

Thus, the main challenge is to access accounting information about the relevant actions of peers in an efficient way[1] such that all relevant information can be retrieved, although it is potentially distributed over the complete p2p system. This is referred to as the *global view*. For example, it is inefficient to query the complete p2p system in order to find all relevant information. Further, due to churn, all information might not be retrievable.

Another challenge is the storage of accounting information in a p2p system. Due to churn, information could be unavailable for a specific period of time or even completely lost. Therefore, a persistent storage

[1] According to (SHL+06) efficiency is understood as the ratio of performance to costs. The most important cost factor in p2p system is the generated traffic. Here, as the required performance is determined, in order to achieve efficiency the costs for accessing accounting information should be as low as possible.

mechanism for accounting information is required. If this mechanisms relies on replication of accounting information to several peers, the consistency of the replicas must also be ensured.

1.3.2 Decentralised Trusted Accounting

Typically in distributed systems in the Internet trust is established by applying a trusted entity, often a central trusted server. This server controls the service sessions in order to ensure that the provided functionality does not deviate from the intended one. This is achieved by taking in to account potential adversaries. Mechanisms that are used in order to ensure system conform behaviour are authentication, authorisation, access control, and further security protocols, e.g. encryption. For an accounting system such a server would collect and store the accounting information and provid it to other functions, e.g. charging.

In p2p systems such a central authority does not exist. Accounting information is collected and stored in a distributed fashion by potentially untrustworthy peers. They may have direct interest in manipulating their own accounting information (e.g., in order to reduce the price of a received service). This is called cheating. Furthermore, peers can collude in order to defraud the system. Hence, in order to achieve trust in a p2p accounting system, two different properties have to be achieved.

First, when collecting the accounting information, it has be ensured that the information is complete and correct. Here, often third party peers are used for observing a transaction; however, due to potential collusion this solution is not fully trustworthy.

Second, while storing the accounting information its integrity and correctness is to be ensured. That is, the information is to be secured against unauthorised access; e.g. a peer should not be able to inject false information or to remove (parts of) it.

1.3.3 Decentralised Cooperation Control

Decentralised rule enforcement faces similar challenges than distributed accounting. Without a central controlling instance in the p2p system a decentralised cooperation control mechanism must be executed by all peers, i.e. each peer must restrict the access to its local resources and services for rule violating peers. Accordingly, decentralised cooperation control consists of three parts: First, each peer has access to the accounting information about all other peers. Second, each peer is able to make a clear decision based on the accounting information, if another peer complied with the rules. Third, each peer must have an interest in restricting the access to its resources and services for rule violating peers. Otherwise, an enforcement mechanism becomes ineffective and will fail to fulfil its purpose. The first part, access to accounting information, was discussed above.

The second part bears the challenge that the accounting information should be clear and easy to evaluate. Based on this, the decision about potential rule violation should be unambiguous.

The third part is the most non-trivial to implement since punishment for granting prohibited access is hard to enforce. This would require that some other entity in the p2p system observed this action, and can clearly decide that it was not in compliance with the rules. However, this cannot be assumed. Therefore, each peer must have a clear incentive for complying with the rules about restricting access to its resources and services. Thus, a peer should feel a loss when it does not comply with the rules about restricting access. An example is a p2p system where cooperation of peers is measured, like a file sharing application. Providing undeserved access to a peer, i.e. uploading a file to a peer not deserving it, should not be measured as cooperation.

1.4 Goal and Objectives of the Dissertation

The goal of this dissertation is *to demonstrate the feasibility of a fully distributed, trusted accounting scheme for p2p systems with intrinsic automatic cooperation control*.

In order to achieve this goal a number of objectives have to be fulfilled. They can be grouped into functional (respectively design) objectives and objectives related to the feasibility and assessment of the developed concepts. The objectives are:

- The design of a fully distributed accounting scheme for peer-to-peer systems.
- The design of a fully decentralised mechanism that allows automatic cooperation control based on the accounting scheme.
- To design fully distributed mechanisms that achieve the trustworthiness of the accounting scheme without having to rely on a single trusted entity in the system.
- Showing the feasibility of the scheme by demonstrating its *viability, trustworthiness, scalability, and efficiency* using analytical assessments as well as simulations.
  - The viability of the accounting scheme with intrinsic cooperation control is shown by designing the required protocols in detail and implementing them in the peer-to-peer simulator "PeerfactSim.KOM".
  - The scheme's trustworthiness is demonstrated by analytical assessments. Values for parameters that configure the accounting scheme's trustworthiness are determined.
  - The scheme's efficiency is demonstrated by evaluating the generated traffic using simulations. The influence of the accounting scheme's parameters on the generated traffic will be analysed.
  - The scalability of the scheme is evaluated by simulating different models of churn and systems sizes.
- Discussing the applicability of the researched accounting mechanism with intrinsic cooperation control in different application scenarios.

1.5 Structure of the Dissertation

The dissertation is structured into seven chapters.

After this introduction, Chapter 2, *"Related Work",* gives basic definitions used within this dissertation and surveys the literature in the three relevant areas. The first part of the literature survey discusses accounting mechanisms in distributed systems. The second part reviews the related work specifically in the area of p2p accounting mechanisms. The third part surveys security mechanisms that use groups of entities to build trust than rather using a central trusted entity.

Chapter 3 *"Token-Based Accounting Principles And Architecture"* introduces the concept, framework, and architecture of the token-based accounting scheme by elaborating the design decisions step-by-step. The resulting building blocks – token structure, transaction protocols, token aggregation, and detection of double spending are described. The building block are an integrated set of mechanism that provide the required functionality and basic trustworthiness.

Chapter 4 *"Relevant Details About Protocols"* describes the details of protocols of the token-based accounting scheme. Strong focus is given in this chapter to the mechanisms that ensure the trustworthiness of the token-based accounting scheme. The protocols ensuring the availability and trustworthiness of accounting information stored in remote account holder sets are described in detail. Further, the concept and organisation of trusted peers is elaborated. A trustworthy aggregation protocol is presented that

implements a trusted token aggregation process. The protocols ensuring the secrecy of the system-wide private key used within token aggregation are described. Also, distribution strategy alternatives to update shares of the key are investigated. Finally, it is discussed how failures within transactions can be recovered.

Chapter 5 *"System Evaluation"* presents the evaluation of the token-based accounting scheme. First, the relevant evaluation criteria and evaluation methodology are presented. Then, values for the accounting scheme's parameters (quorum size and account holder set size) required to achieve trustworthiness are determined using analytical and simulation studies. The third part presents a message-based traffic analysis of the most relevant protocols. The fourth part presents the simulation model and simulation results about the traffic generated by the token-based accounting scheme. After this the simulation model and simulation results for system key management are presented. Finally, the traffic generated by the token-based accounting scheme is compared to the most relevant related work.

Chapter 6 *"Deployment Issues"* deals with issues that arise when the token-based accounting scheme is applied in practise. It describes bootstrapping strategies and discusses application examples for commercial scenarios as well as a file sharing scenario.

Chapter 7 concludes this dissertation. It summarises the results and outlines the contributions to accountability in peer-to-peer systems and the application of distributed security mechanisms in peer-to-peer systems. Furthermore, an outlook to future work resulting from the dissertation's research is given.

Subsequently, the appendices to the dissertation are presented.

2 Related Work

In this chapter the related work relevant for the goal of this dissertation – development of an accounting system with intrinsic cooperation control in autonomous distributed systems – is summarised. As stated in Section 1.3 the trustworthiness of the accounting records created by such a system is a core requirement. Accordingly, the related work is structured into the two parts. First, relevant accounting mechanisms in computer networks are presented with focus on accounting mechanisms for peer-to-peer systems. Then, security schemes designed for distributed systems are discussed, since such mechanisms are applied in the token-based accounting mechanism.

2.1 Definitions in Context

Within this dissertation the main concepts are peer-to-peer, accounting, and cooperation control. In order to establish a common understanding throughout this dissertation, they have to be defined in more detail.

2.1.1 Definition of Peer-to-Peer

Peer-to-peer is a communication paradigm in the Internet. The term peer-to-peer (p2p) was created in the year 2000. According to an initial definition given by Clay Shirky in (Shi01) p2p is "*a class of applications that takes advantage of resources - storage, cycles, content, human presence - available at the edges of the Internet. Because accessing these decentralised resources means operating in an environment of unstable connectivity and unpredictable IP addresses, peer-to-peer nodes must operate outside the DNS and have significant or total autonomy from central servers.*" Shirky also provides a litmus test, according to which an application is peer-to-peer if (1) it allows for variable connectivity and temporary network addresses and (2) it gives the nodes at the edges of the network significant autonomy.

The IRTF Peer-to-Peer Research Group applies two definitions: in its charter the group defines p2p as:

"... a way of structuring distributed applications such that the individual nodes have symmetric roles. Rather than being divided into clients and servers each with quite distinct roles (such as Web clients vs. Web servers), in p2p applications a node may act as both a client and a server. p2p systems are in general deployable in an ad-hoc fashion, without requiring centralised management or control. They can be highly autonomous, and can lend themselves to anonymity."

Additionally, in (RM06), a *p2p network* is defined to exhibit the three characteristics self organisation, symmetric communication, and distributed control. This definition is taken from (RBR+04). Furthermore, a self organising p2p network is defined according to (RD01) as network that "automatically adapts to the arrival, departure, and failure of nodes."

The term "p2p network" is somewhat confusing as also overlay networks are a very important part of p2p. Accordingly, Mahlmann and Schindelhauer (MS07) state that the research community has agreed roughly that a p2p network is an overlay network where the participating computers communicate as peers without central coordination in order to provide an application jointly.

In this dissertation, in order to distinguish the *overlay network* from a system that fulfils the p2p definition and provides additional functionality than an overlay network the term *p2p system* will be used.

The peer-to-peer working group of the Internet2 consortium defined *p2p computing* as *"... sharing of computer resources and services by direct exchange between systems. These resources and services include the exchange of information, processing cycles, cache storage, and disk storage for files."* (see (SF02)). The working group seems to be inactive since April 2004. In a similar way Barkai (Bar01) defined p2p computing as "*... a network based computing model for applications where computers share resources via direct exchanges between participating computers*."

In research, several people have also defined p2p systems. According to Yang and Garcia-Molina (YGM02) "*Peer-to-peer (p2p) systems are distributed systems in which nodes of equal roles and capabilities exchange information and services directly with each other.*" Aberer and Hauswirth (AH02) extract from Shirky's definition three underlying principles: The principle of sharing resources, the principle of decentralisation, and the principle of self organisation.

In a later work, Hauswirth (HD05) modifies this definition by ascribing the following characteristics to p2p systems: symmetry of roles, decentralisation (no central coordination, no central data base, no peer has a global view of the system), self organisation, autonomy of peers, unreliability of peers and network links, availability of data stored in the system in the presence of distributed storage, unreliability, and unknown trustworthiness of peers, etc.

Summarising all these characteristics, one of the most complete definitions was composed by Steinmetz and Wehrle in (SW04, SW05a) where they summarise nine important characteristics of p2p systems. This definition will be applied in this dissertation.

Steinmetz and Wehrle first gave as a basic definition of p2p systems (SW04): *A Peer-to-Peer system is a self-organising system of equal, autonomous entities (peers) aims for the shared usage of distributed resources in a networked environment avoiding central services.*

Further, they group the nine properties that characterise p2p systems into the two categories - *decentralised resources usage* and *decentralised self-organisation* (though a single system rarely exhibits all of these properties).

- Decentralised Resource Usage:
  - Resources of interest (bandwidth, storage, processing power) are used in a manner as equally distributed as possible and are located at the edges of the network, close to the peers.
  - Within a set of peers, each utilises the resources provided by other peers. The most prominent examples for such resources are storage and processing capacity. Other possible resources are connectivity, human presence, and geographic proximity.
  - Peers are interconnected through a network and in most cases distributed globally.
  - In order to deal with variable connectivity of peers p2p systems to introduce new address and name spaces above of the traditional Internet address level. Hence, content is usually addressed through unstructured identifiers derived from the content with a hash function. Consequently, data is no longer addressed by location (the address of the server) but by the data itself.
- Decentralised Self-Organisation:
  - In order to utilise shared resources, peers interact directly with each other. In general, this interaction is achieved without any central control or coordination.
  - Peers directly access and exchange the shared resources they utilise without a centralised service.
  - In a p2p system, peers can act both as clients and servers
  - Peers are equal partners with symmetric functionality. Each peer is fully autonomous regarding its respective resources.

– Ideally, resources can be located without any central entity or service.

A wide overview over the state-of-the-art in p2p research gives e.g. (SW05b).

2.1.2 Definition of Accounting

Accounting is the mechanism (of an information system) that provides accountability. There exist several definitions for accountability in information systems (IS).

In (Car08) it is defined as "*... the ability of a system to keep track of who or what accessed and/or made changes to the system.*"

In (Nat99) accountability is defined as "*... the process of tracing IS activities to a responsible source.*"

Both definitions agree and express the meaning of accountability used in this dissertation. Important to note is that in p2p systems, not all IS activities will be accounted for. The size and potentially numerous activities in p2p system require limiting accounting to relevant IS activities. Relevance is defined by the application provider, and depends on the application scenario and the goal the application provider hopes to achieve using an accounting functionality. In this dissertation, in accordance with the above definitions, the following definition is used:

Accounting is the process of tracing ***relevant*** *IS activities to a responsible source. Relevance is defined by the application applying accounting.*

Accordingly, accounting is a mechanism that builds the foundation for other mechanisms that use accounting information as input. Example mechanisms are charging and billing.

Further, accounting has to be distinguished from incentive systems. An incentive system gives incentives to peers for stronger cooperation. It may use accounting information in order to measure the peers' contribution, however different alternatives exist and most incentive systems use considerably less detailed information than data an accounting system conventionally provides. Example incentive systems are upload traffic measurement in KaZaA (Sha), bandwidth allocation in eMule (eMu04a), or the use of virtual currencies like MojoNation (Wik08b).

An accounting system is not a virtual currency system. Currencies are typically impersonal, in the sense that a unit of currency does not contain information about ownership, usage, etc. Furthermore, currency systems (as well as virtual currency systems) are complex systems with mechanisms like management of money supply. Therefore, accounting in information systems and virtual currencies are separate mechanisms.

2.1.3 Definition of Cooperation Control

p2p systems are built in cooperation with the participating peers. Therefore, it is meaningful to define rules about cooperation with the goal of increasing the system's performance and achieving fairness. According to the "Carrot and Stick" approach (Gal02) presented above, cooperation control consists of three parts: definition of rules, compliance check, and infliction of consequences for violating the rules. This will be described using the following example.

Known rules for file sharing applications exist, for example, to provide a specific upload/download bandwidth or traffic ratio. For example, KaZaA implements such a rule (Sha).

When rules are defined, the compliance with the rules must be controlled. Otherwise it cannot be determined if peers follow the rules. The basic data to inform this decision is accounting information. According to the example, KaZaA, the p2p file sharing client measures the upload traffic the peer provides (Sha).

There is no need to behave in compliance with rules if there are no consequences for peers that do not follow the rules. So the KaZaA-client will limit the available download bandwidth if the peer does not contribute enough upload bandwidth (Sha).

In conclusion, the rules contain a mechanism for controlling compliance, and a mechanism for enforcing rules by inflicting consequences on peers that do not follow the rules. Within this dissertation, cooperation control consists of these three parts: definition of rules of cooperation, control of compliance with rules using accounting information, and infliction of consequences for violating the rules.

2.2 Accounting in Distributed Systems

Before reviewing the related work in the context of p2p systems, this section discusses the how accounting functionality is implemented more generally in distributed systems (CDK02).

In the context of implementing quality of service in computer networks, it is discussed how network providers can implement different quality of service classes (see, e.g., (Sch00)), and charge for them using a signalling and charging architecture (see, e.g., (SFPW98b, SFPW98a, SBGP99, Kar00, BDH+03, KRS03, SRG+03)). These mechanisms are not applicable to the p2p domain, since accounting is performed on the transport layer. Similar, work in the area of accounting in UMTS and WLAN networks such as (RFS07) cannot be applied. Accordingly, this section discusses accounting mechanisms implemented on the application layer in centralised and distributed systems.

First, accounting in distributed systems research in general is reviewed. Then, two special instances of client/server systems is given special attention; these are Content Distribution Networks (CDNs) and Radius. Then, accounting mechanisms for Grids and micropayment schemes are reviewed for concepts that can be transferred to p2p systems.

2.2.1 Accounting in Distributed Systems Research

In distributed systems research, authorisation has received a lot of attention. However, there are only very few papers also concerned with distributed accounting mechanisms for distributed systems.

In (Var91), Varley summarises the requirements and issues of accounting management in distributed systems. In (Neu93), Neuman describes a model for authentication, authorisation, and accounting that is based on so-called restricted proxies. A proxy is a token that grants rights and privileges to the owner. Authorisation and accounting are interdependent. Accounting depends on authorisation in order to control the transfer of resources from one account to another. Authorisation requires from accounting a verification for allocated resources. Accounts are maintained at accounting servers. Each account holds an access control list and accounting records each for a different resource type or currency type. Clients may have their accounts located at different accounting servers. The basic concept uses quotas, which are implemented by transferring funds of the appropriate currency out of a client's account when resources are allocated to the client. When the user releases the resources, the funds are transferred back into its account. If a client C requests resources from a server S, C sends a signed check to S, stating the requested resources and the resource provider, as well as a unique check identification number. This way C cannot re-use the same check in order to double spend. When C's request is completed, S endorses the check, and deposits it with its accounting server $\$S$. If C uses a different accounting server, $\$S$ forwards the check to C's accounting server $\$C$. As these messages are out-of-band to the resource usage, accounting servers also deal with checks returned due to insufficient funds, forged checks, or misdrawn checks.

Summary

Neuman's work comes close to the requirements of an accounting mechanism for p2p systems. Although in the presented system, accounting does more then recording the usage of resources - it also takes care of payment for resource usage. Accordingly, there is also a mechanism present to deny access

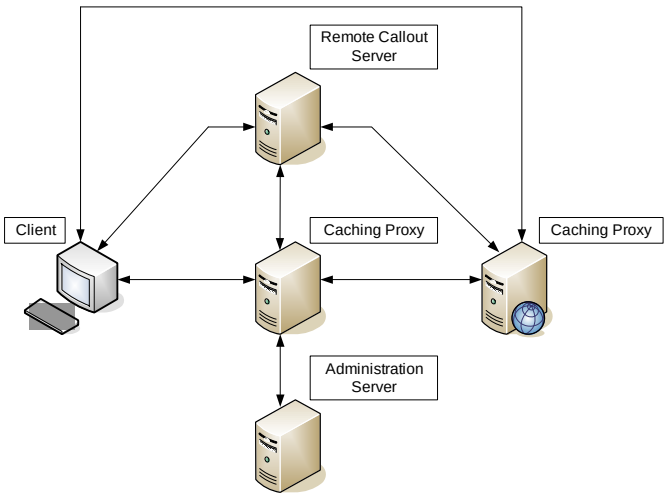


Figure 2.1: Components of CDNs (cf. (BEH+01))

to further resources for clients that do not have enough currency available. However, Neuman assumes the existence of trusted accounting servers. This cannot be assumed within p2p systems.

2.2.2 Accounting In Client/Server Systems

Accounting has never been a major research issue in client/server systems. The reason is that accounting in such systems was intuitively easy, as two clients never communicate directly, so all communication is controlled by the server. Therefore, the server can log every access activity of the clients and create trustworthy accounting records. In order to show accounting solutions in more complex client/server based systems, accounting in content distribution networks (CDN) and in Radius (RAD00a) is discussed.

2.2.2.1 Accounting in CDNs

Accounting in Content Distribution Networks (CDNs) is described in (TOC+00). The components of a CDN are depicted in Figure 2.1. Accounting in CDNs has the primary purpose of accounting for proxylet functions in order to allow for the billing of services that are used by clients or publishing servers. It is required at hosts that act as remote callout servers. As a secondary purpose, accounting information is used to aid in operations, billing, and SLA verification. Accounting information typically is provided in the form of log files created and stored at a server, either in aggregated or in detailed form. Accordingly, all servers of a CDN are trusted entities.

Accounting information is collected by the remote callout server and the caching proxy server. If there is a central administrative server (administrative server model), both the remote callout server and the caching proxy server will send collected accounting information to the administrative server. The remote callout server might exist in a different domain than the administrative server. Therefore, mutual authentication between these servers is required. It is assumed that the accounting information is collected and stored in a secure and trustworthy manner at these servers.

When CDNs are interconnected, different interconnection agreements can be applied. Independent any interconnection agreement, accounting information from foreign domains must also be exchanged. This requires a generalised standard process that supports different current and potential future business models.

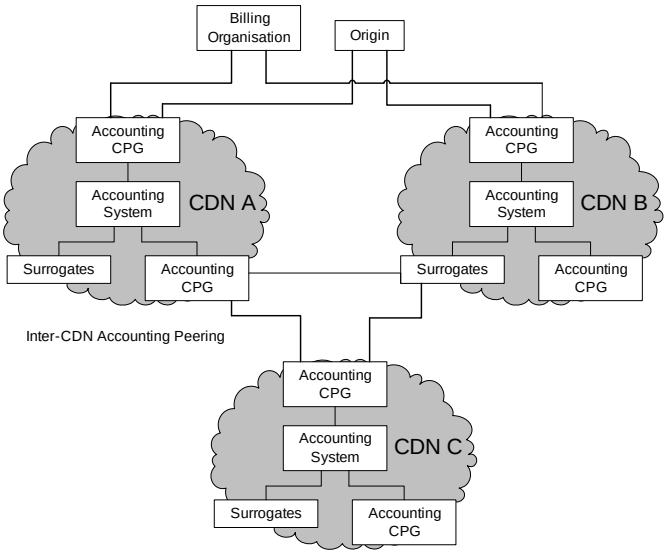


Figure 2.2: CDN accounting peering architecture (cf. (BEH+01))

In CDN peering, so called Accounting CDN Peering Gateways (CPGs) exchange accounting data collected by the internal accounting servers. In order to achieve scalability, CPGs can aggregate accounting information before it is transferred. Figure 2.2 depicts such a CDN peering system.

Another approach to authentication, authorisation, and accounting for CDNs is suggested in (CLM03). Here, accounting is performed by a central server. Accounting peering is not considered.

Summary

In summary, accounting in CDNs is not concerned with secure accounting information collection, but with secure and authenticated access to accounting information from different domains. A different set of issues must be addressed than those in decentralised autonomous systems like p2p.

2.2.2.2 Accounting in Radius

The Remote Authentication Dial In User Service (RADIUS) standard describes its accounting functionality in (RAD00b). RADIUS implements the function triple authentication[1], authorisation[2] and accounting (AAA). The purpose of accounting is to provide data for billing users and generating statistics.

The RADIUS accounting protocol delivers accounting information from the Network Access Server (NAS) to a RADIUS accounting server. The basic concept is that the client computer sends at the beginning of a service delivery an Accounting Start packet to the RADIUS accounting server. The packet describes the type of service being delivered, and the user service is being delivered to. The accounting server responds with an acknowledgement. At the end of the service delivery, the client will send an Accounting Stop packet to the Radius accounting server. This packet describes the type of service that was delivered and optionally statistics, such as service duration, and input and output traffic. Again the accounting server responds with an acknowledgement.

All these tasks are executed by a centralised Radius server. There are mechanisms that can redirect the accounting traffic. This is used, for example, when the accounting server fails.

[1] Authentication: the process of establishing the digital identity of one entity to another entity (RAD00a).

[2] Authorisation: the process of the granting specific types of privileges (including "no privilege") to an entity or a user, based on their authentication (RAD00a).

Summary

RADIUS accounting is fully centralised and relies on trusted servers. Hence, its concept cannot be transferred to the fulfilling the objectives of this dissertation.

2.2.3 Accounting in Grids

In the Grid community, accounting for provided services was first considered in (THMA02), followed by (BB03).

Thigpen et al. suggest in (THMA02) an "accountless accounting" method for Grid environments, because local user accounts are not feasible. Their concept suggests that each Grid entity autonomously sets rates for the use of its resources. Consumers have to bid for resources and it is envisioned that a market for resources develops. When a service is provided, a receipt for the consumed resources is created by the supplier and sent to the consumer. In order to build the market, each participating site should try to maintain a zero balance. In order to calculate a balance, each site uses a formula the weighs the different resources in order to aggregate them to a uniform resource. Sites that cannot provide resources themselves should establish a partnership with resource providing sites in order to achieve a zero balance. In summary, the whole concept is still abstract and an accounting functionality has not been explicitly designed.

Barmouta and Buyya present in (BB03) the concept of Gridbank, which is an accounting and micro-payment handling system. It is thought to establish an economy with a Grid environment, as well as for usage in e-commerce applications. GridBank is designed as just another resource provider in the Grid. Both consumers and suppliers in a Grid environment can open an account with GridBank. There is a Grid Trading Service or a Grid Market Directory that establishes prices for resources. When a provider requests a resource from a supplier, either the consumer informs the supplier with its account details, along with the resource request, or the provider uses a GridCheque purchased at GridBank. During the resource provisioning, the Grid Resource Meter measures the actual resource usage. After the job is finished, the providers Charging Module contacts GridBank and requests a transfer of funds. It also send an accounting record with the request. GridBank is designed as a client/server architecture. In summary, GridBank sets a focus on how to build a market in a Grid environment, where many different kinds of resources are offered. It uses a centralised, trustable bank, which is not applicable to p2p systems.

Another bank-based approach to accounting in Grid environments is presented in (SGE+04). It also uses a centralised, trustable bank. In order to achieve scalability, accounts can be distributed across several bank instances. Similar to the previous concept, this one is also not applicable to p2p systems.

In (YLLH04), an economy-based accounting system for Grid environments is presented, that is it uses a centralised accounting server and set a focus on resource pricing.

Göhner and Rückemann present in (GR06) the concept of accounting in Grid environments with a detailed monitoring concept, in order to create accounting records. Monitoring is performed as a decentralised function, however, accounts and accounting data are stored at accounting servers, which are assumed to be trusted. In (GWG+07), Göhner et. al. present the concept of an accounting model for dynamic virtual organisations in the Grid that focuses on economic and business aspects.

An architecture for secure resource peering is SHARP (FCC+03). SHARP was not directly designed for Grid, but for Grid-like distributed resource sharing systems. SHARP is used, for example, in PlanetLab. SHARP uses a ticket system to book resources. Tickets are issued by resource providers by signing the ticket with the resource consumer's public key. Ticket owners can also delegate part of the resources to other users. Also, similar to airlines, overbooking is supported in order to facilitate better resource usage.However, there is no mechanism that ensures global fairness of resource usage. All mechanisms are meant to ensure that a site is not overbooked, potentially by forged ticket. Accordingly, there is no global accounting mechanism existing in SHARP.

Charging for Grid services based on accounting information is, for example, presented by Stiller in (SGF+01).

Summary

In conclusion, accounting in Grid environments has some similarities to p2p environments. Services are supplied and consumed in a decentralised manner. For accounting, the concepts differ in their monitoring styles. Some concepts use explicit monitoring services provided as Grid services, other concepts use decentralised monitoring. Accounts for participating sites in a Grid are always hosted on centralised servers. In the context of trust, authentication is observed, however monitoring and account holding instances are assumed to be trustworthy. Therefore, accounting solutions for Grid environments cannot be transferred to p2p systems.

2.2.4 Micropayment Systems

There exist a wide variety of micropayment and macropayment systems for the Internet; to discuss them is beyond the scope of this dissertation. An overview can be found at (Pei01). Typically, these systems were designed to accomplish small payments over the Internet, e.g., paying services that are worth a few Euros in the case of macropayment systems or fraction of a cent in the case of micropayment systems. Because these systems have a counter value in a real world currency, these systems require a high level of security. Therefore, most of them are designed as a client/server system, where the server is well protected and trusted. Two basic types of micro/macropayments can be distinguished. One system uses only accounts hosted at the server. If a payment is to be made, funds are transferred from one account to another. Accordingly, the server must be reachable for all participants anytime. This solution is typically used for macropayments, because the effort is too expensive for micropayments. The second type uses anonymous objects, like electronic coins, that are issued by the central server. The objects are anonymous and can be handed over to another participant. They behave like a coin or bank note in the real world. In order to achieve security in such a system, the objects are enriched with cryptographic mechanisms that enable them to be traced in order to detect double spending (see e.g. (CFN90)). Therefore, these coins grow in size when they are handed over to another peer, due to the added information.

Summary

For accounting in p2p systems these systems are not adequate for several reasons. First, micropayments are anonymous. This is a basic characteristic of currency. Secondly, accounting information cannot easily be collected and stored. Third, these systems are highly centralised in order to build a central trusted authority to give the systems trustworthiness in the Internet environment.

2.2.5 Summary

Accounting in distributed systems typically relies on trusted, central servers; in most systems it is one single accounting server used. This assumption is contrary to the concept to p2p systems. Hence, accounting concepts from these system cannot be transferred to build an accounting scheme for p2p systems. Grid environments have some similarities to p2p environments. Here, distributed monitoring is applied to collect accounting information. This is a concept that could be transferred to p2p systems. However, the collected information is stored on trusted servers.

2.3 Accounting in Peer-to-Peer Systems

In peer-to-peer systems, the topic of accounting has been often discussed in context of other topics, especially incentives and preventing free-riding. These works require the collection and storage of accounting related information. However, the focus of most of these papers is the correct design of incentives (see e.g. (ACM04, ACS05)), and not the design of an efficient and trustworthy accounting scheme, which is an objective of this dissertation.

2.3.1 Classification of Peer-to-Peer Accounting Schemes

This section outlines the different design parameters for building an accounting system that meets the described requirements. The problem can be structured into two main parts: *Type of information collection* covers the options as to how and in what form information is collected. *Information storage location* concerns the options of where the collected information can be stored.

2.3.1.1 Type of Information Collection

In p2p systems, there are few options as to where accounting information can be collected. Assuming real p2p connections (without a third peer in the middle as a "trusted party"), the information can only be collected at the service provider peer, at the service receiver peer, or at both peers. This collection process usually consists of a metering process and an evaluation process. The metering process measures the used bandwidth (the amount of data received/sent), time (the duration of the service), or some service specific signal like "file complete". The evaluation process interprets this information to generate accounting events. These events determine when an accounting record is created.

Accounting records contain accounting information and can take several forms.

Pure Numbers: The simplest form is pure numbers. For every peer, there exists an account balance stored somewhere in the p2p systems. For example, for each MByte uploaded, the peer's account balance is increased by one. Pure numbers can be changed very easily. Therefore, this kind of accounting record is especially vulnerable to fraud. Nevertheless, this form of information consumes very little space and bandwidth when it is transferred between peers.

Receipts: Receipts are documents of fixed form that can contain all kinds of accounting information, e.g., transaction partners, transaction object, metering information, and an evaluation of it. This evaluation could be the information about how this receipt modifies the peer's account balance. This part of information would be similar to pure numbers. In comparison with pure numbers, the additional information of a receipt adds some trustworthiness to the accounting information, because more information is available, which could help to detect fraud.

Signed Receipts: Signed receipts are normal receipts with a signature to add integrity to the information contained in the receipt. This is a very important step to achieve trustworthiness of information. Additionally, a signature authenticates a receipt. Working with signed receipts requires that every peer own a private/public key pair. Such a key pair can, for example, be issued by a central authority, or peers could create the key pair themselves and using Public Key Infrastructure (PKI) the keys would be certified. For example, JXTA (Sun04) uses self-signed certificates. The disadvantage to this approach is that the keys are not persistently associated with a peer. A peer can easily create a new key pair if it wants to revoke its identification. This problem can be ameliorated by using a Web of Trust as implemented by PGP (Sch96) or the cryptoID-project of JXTA (JXT04). A central authority implementing a PKI does not have this disadvantage, though its usage seems contrary to the p2p paradigm. Nevertheless, the responsibilities of a central authority within the p2p system could be reduced to issuing certificates with

keys, and approving the validity of keys. Apart from these tasks all the other parts of a p2p system could still be delivered according to the p2p paradigm.

Receipts are normally signed by the author of the contained information. Indeed, receipts could also be signed by the transaction partner to include an agreement about the information in the receipt. This can also be accomplished by bilaterally signing a receipt. However, one should bear in mind that each signature increases the size of the receipt. And size matters for the efficiency of the accounting system in terms of created overhead. Depending on the cryptography mechanism used, the size of the signature could be large. For example, today's RSA signatures are typically found to be not smaller than 1024 bits.

The information types *pure numbers*, *receipts*, and *signed receipts* are aggregated in Figure 2.3 into the term *receipts*, because these are not issued. The following information types have to be issued before they can be used for accounting.

Tokens: In the context of this chapter the term *token* is used for any kind of issued document. An issuer releases a specific number of these documents (tokens). Thus, the number of tokens available in a p2p system can be limited. This results in a specific characteristic that is otherwise hard to achieve - through issuing a limited number of tokens, they become a scarce resource. Accordingly, tokens can represent other scarce resources. However, they are not as easy to handle as normal receipts, due to two major problems - *Forgery* and *Double Spending*. With respect to double spending, a token must be clearly identifiable, and must therefore contain a unique identifier. Also, there must be a mechanism built into an accounting system to check for double spending. To avoid forgery, it must be ensured that only the issuing authority can create a token and therefore, a token must be signed by the issuing authority with its private key.

There are several options for the token issuing authority in an accounting system. For example, in digital cash systems, a central bank issues the tokens (e.g. (Sch98)). This option ensures trustworthiness and control, but it does not conform to the p2p paradigm. In (YGM03), a micropayment scheme for p2p systems is presented that relieves the central bank many of its responsibilities. The second option is that each peer issues its own tokens. Here tokens are very similar to signed receipts. It is notably difficult to control the number of tokens issued by every peer. Therefore, determining the economic value of a peer's token is difficult, too. The third option is a hybrid of the first and second concepts. It is issuing tokens by a subgroup of peers, and controlling the subgroup's actions in order to control the total number of tokens in the system.

Token-based accounting systems can further be distinguished by the usage of the tokens. Tokens can either be used as a kind of micropayment system or as receipt, i.e. a token represents a very specific part of a provided/consumed service (e.g. 1 MByte data transfer). Tokens become receipts by adding accounting information to the token. Thus, a token can represent all kinds of transactions or parts of transactions.

Proof of Work: Proof of Work (PoW) is another micropayment concept where a user has to show that he performed some computationally difficult problem to be eligible to receive a service (DFM01). Nevertheless, PoW cannot be redeemed by the receiver for something of value to him. Therefore, they can be used to avoid denial of service attacks or flooding attacks. Further, they do not present an economic measure (DFM01). Also, to create a PoW, limited CPU cycles that could be traded in the p2p system (and will then represent a scarce resource themselves) are needed.

In (RS96) two micropayment systems based on Proof of Works are presented. In both systems, each participant issues his own secrets. As previously mentioned, using such systems poses a problem for users in determining the economic value of another user's virtual currency. In order to overcome the need to identify the exchange rate of two users' virtual currency in (JJ99) Proof of Work-based systems are extended to work with a central broker that issues secrets centrally. This way only one virtual currency exists in the system.

Further, for accounting PoW can be seen as a special form of receipt or self issued token. Further concepts about storing used PoW are not described. Thus, PoW in general can be regarded as improper for accounting systems.

Summary

Micropayment systems as well as Proof of Work systems are not applicable to accounting mechanisms.

Tokens have some advantages and some disadvantages of receipts. Receipts are created by each peer. Therefore, their number is unlimited. Tokens are issued; here an additional mechanism is available that can be used to limit number of tokens the available to a peer. This could be used, e.g., to limit access of peers to services and resources. The disadvantage of tokens is the additional overhead created by the issuing process. Also due to the issuing information tokens are larger in size than tokens, which enhances the traffic overhead of the accounting system.

2.3.1.2 Information Storage Location

In whichever form accounting data is collected, it must be stored at some place in the network - an account. The account location is an important design parameter because it strongly influences the traffic overhead of an accounting system. For every transaction, one or more accounting records accumulate and must be transferred to the account. Also, the location influences how easy it is for a peer to manipulate accounting records and defraud the system. There are three basic alternatives for the account location: accounts can be located locally at the peer that collects the data, at a central server, or remotely at a peer other than the collecting peer. Centrally held accounts obviously are contrary to the p2p paradigm. For every transaction, communication with the central entity would be necessary to transfer the record. Therefore, this solution is out of the question. The advantages and disadvantages of the two other alternatives will now be elaborated.

Local Accounts : Storing accounting records at the place where they accrue has the obvious advantage of reduced traffic, because the records do not need to be sent to the account holder. However, using local accounts poses an important trust problem. Using pure numbers or self-signed receipts to store information enables users to easily change the contained information in order to defraud the accounting system. The appearance of KaZaA Lite as competitor to KaZaA is an example to this behaviour (Wik08a). [3] Therefore, receipts should be signed either by the transaction partner or a third party. Alternatively, tokens could be used. With tokens, accounting records need to be transferred between the transaction partners. Therefore, in terms of bandwidth usage, storing accounting records locally is only advantageous in comparison to other storage location, if the transaction partner needs to get the records for some reason. For example, both transaction partners have to agree on the contained accounting information. Further advantages are that users have immediate control over the collected accounting records. No redundancy need be built into the system, because a peer's accounting records are not needed when it is offline. Also, users are themselves responsible for backing up their data.

Remote Accounts: The alternative location for storing accounting records is third peers. Using third peers, hence separating account holders from account owners, clearly makes it more difficult for any one peer to fraudulently manipulate accounting records. Depending on the p2p application's requirements, special mechanisms to ensure information integrity such as signing might not be needed.

However, using remote accounts requires the exchange of more administrative messages between peers. This stems from the need for redundancy. Because the account holder is not always available, several account holders per peer, each holding a replication of the account, are required. All replications of an account need to be kept consistent. Therefore, mechanisms to detect potential inconsistencies as well as mechanisms for determining the actual account status in a situation of disagreement are required. A Byzantine quorum (MR97) is an example of a mechanism that might be used in that case.

[3] KaZaA limits a peer's download bandwidth according to its ratio of performed upload to received download. Because the download limit is local rule based on local accounts, users can also run a modified application, e.g., KaZaA Lite, that does not collect accounting information and does not enforce the rule.

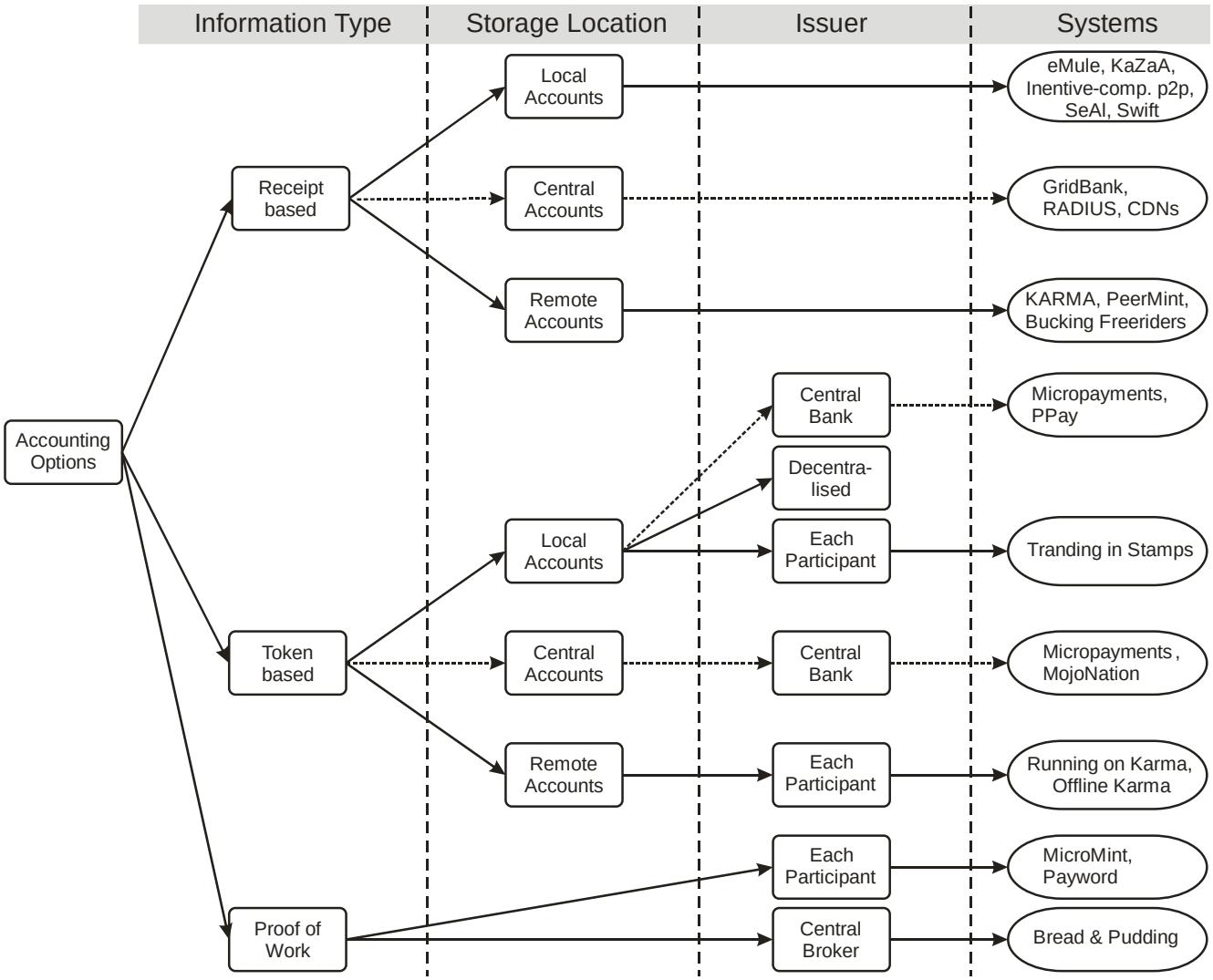


Figure 2.3: Classification of accounting systems for autonomous distributed systems

Summary

Local accounts are to be preferred to remote accounts, because they do not involve a third party during transactions in order to generate and store the accounting information. However, local accounts have a trust issue. A peer storing accounting information about its own actions has a strong incentive falsifying this information. Thus, cryptographic mechanisms need to be applied in order to ensure integrity of accounting information. Still, peers could omit or erase the latest added part of information by not updating the account information or reverting it to an earlier version of the account. Thus only such information should be stored locally, where a peer does not have this incentive.

On the other hand remote accounts also do not offer full trustworthiness, as remote accounts can be attacked by an adversary. Also collusion of the third party peers storing an account and the account owner is possible. Thus, if remote accounts are used further mechanisms are required that ensure the security of the stored information.

Figure 2.3 depicts this classification of p2p accounting systems and maps existing p2p accounting solutions to the classes. These solutions are presented now. Solutions with central information storage have been discussed above.

2.3.2 Existing P2P Accounting Systems

Today, most mechanisms that accomplish the task of accounting are built into systems that have an objective that requires accounting. Most of these systems are incentive systems or economic frameworks that aim at preventing free-riding or at establishing a market for resources or services in p2p systems. In the following, the related work that implements some form of accounting according to the classification is discussed. A work that presents a general software architecture that can be used to implement many different kinds of accounting schemes is presented in (HGS05).

2.3.2.1 Systems Without Receipts Using Local Accounts

File Sharing Applications "KaZaA" And "eMule"

Currently the two most widely used p2p accounting mechanisms are KaZaA's participation level (Sha) and eMule's credit system (eMu04b). Both systems account for the amount of data uploaded and downloaded, and store the collected information locally in form of a pure numbers. KaZaA's system uses the ratio of uploads to downloads (measured in amount of data transferred) to calculate a peer's actual maximal allowed download speed. The higher this ratio is, the higher the maximal allowed download speed. This is a typical incentive mechanism for file sharing systems. However, KaZaA's system is easy to cheat because the accounting information is stored locally. In fact in the KaZaA clone KaZaA Lite (Wik08a), the participation level is removed. In contrast, eMule's credit system is used to determine a requester's position in the provider's download queue. The position is determined by the amount of data the requester uploaded to the provider. This system has the obvious advantage, that it cannot be cheated. The provider keeps his accounting records that only influence his own behaviour. The disadvantage is that this system only accounts for local observations. A peer could have uploaded to the system much more than it downloaded. However, if it downloads from a peer to which it did not upload before, it will receive an unfavourable position in the provider's download queue.

Summary: In sum, using receipts stored in local accounts has the disadvantage, that it is only applicable to local accounting information; that is, a peer records the actions of other peers with itself. This information is only used for local system administration. Gobally collected accounting information cannot be stored in a trusted way in local accounts, because peers could always chose to not update a local account or to even erase it, which results in whitewashing.

Swift

Another system that uses local accounts to store pure numbers as accounting information is Swift (TPM04). In contrast to the previously mentioned systems , it is not used in practise yet. Swift basically is a behaviour model for p2p file sharing to support fair and fast large scale distribution of files. Each peer maintains a credit for every other peer it is connected to. A peer will only upload to a peer with a positive credit balance. Because the accounting data only affects the local peer behaviour, peers have no incentive to falsify the collected information.

Summary: Swift's basic concept is similar to eMule, however its participation rules are more complex. Accordingly, it does not provide a global view about peer's actions.

Incentive-compatible P2P Systems

In (NNS+04), Ngan et al. present two systems that use local accounts for storing accounting data. The schemes include incentives for storing fraudulent entries in accounts. This is achieved by the auditing of accounts by other peers.

The first system is a p2p remote file storage system. Each peer keeps records about the foreign files it stores locally in a "local list", and about its own stored files in the p2p system in a "remote list". A peer B storing a file for peer A can audit A's remote list to check if A lists that storage. If not, B

can remove the file. Accordingly, A has no incentive to lie about the services it receives from the p2p system, because then the service would be stopped. The second kind of audit checks A's local list - if the mentioned remote peer stores the corresponding record in its remote list. If fraud is detected the auditor can present the case to all other nodes storing objects for A. They would then delete A's files. If this audit is reliable, peers do not have an incentive to commit fraud.

The other suggested system is used for trading bandwidth in a file sharing scenario. Peers store locally a list of peers they have shared files with - how much a peer uploaded to, and downloaded from that peer. Peers can reject sharing requests if the requester has already a negative bandwidth balance with that peer. In order to request files from a peer that is unwilling to serve due to a negative balance, a debt path is created. In the debt path each peer has credit with the next peer. Then the requested file is split into fixed size blocks. Now the requester then requests specific blocks from the provider while the request messages are sent along the debt path. The requested block is sent directly from the provider to the requester, however the credit of each individual peer is adapted according to the requests sent over the debt path. Accordingly, peers have an incentive in helping to maintain a dept path.

Summary: In sum, both systems are very specific use cases. The two very different concepts show that it is hard to transfer them to other scenarios. Furthermore, they concepts are not applicable to scenarios that require trustworthy accounting records, as malicious behaviour is not considered.

SeAl

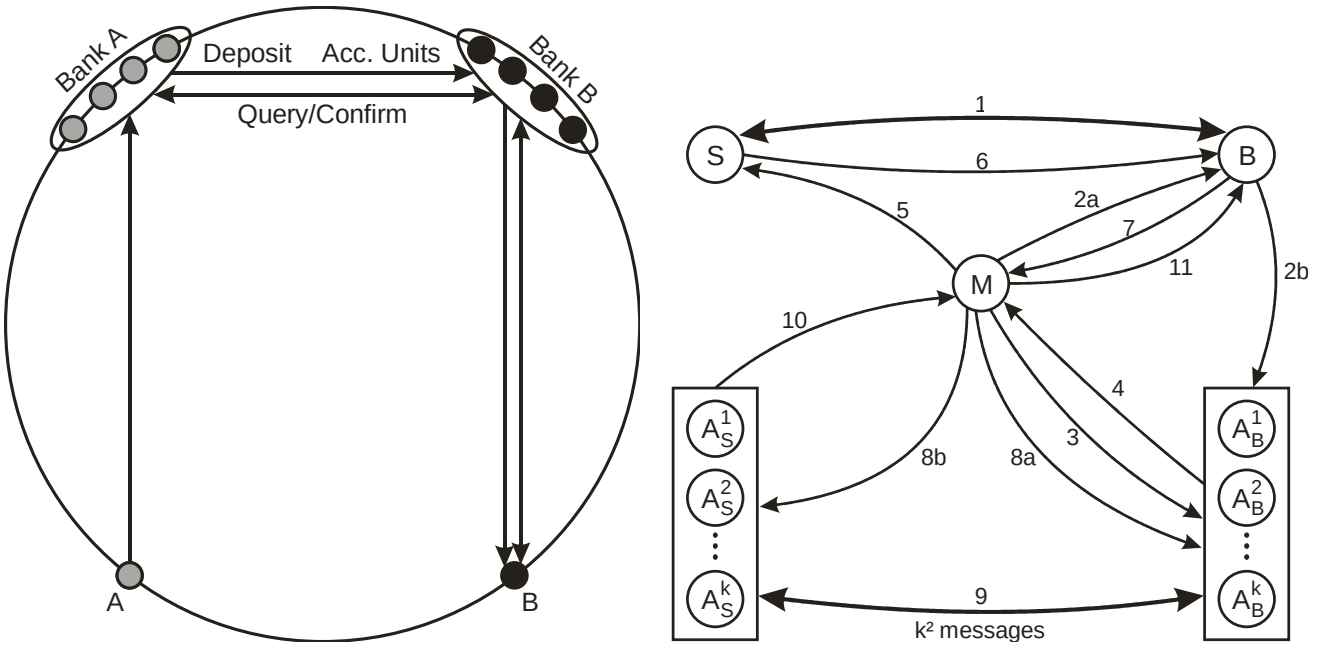
SeAl (NT04) aims at resolving the free-riding problem in p2p systems. Its basic means of accounting information are so-called "favours". When a consumer receives a service from a provider, both peers store locally a favour about the transaction. A favour is a mutually signed transaction receipt stating the consumer, the provider, the value of the transaction, and a time stamp. Each peer posses a private/public key pair. The time stamp is used to devalue a favour over time. Favours have to be redeemed. If a peer A receives a service request it can reroute the request to a peer B that owes A a favour.

SeAl employs a black list to publish selfish behaviour. When a peer does not return a favour when this is requested, it is considered selfish behaviour. Also being offline is considered selfish behaviour. The black list is hosted in a distributed hash table (DHT). The black list identifier for the DHT is a hash value of the String "BLR" concatenated with the peer ID of the black listed peer. The black list stores transaction receipts of transactions where the peer acted selfish. When a peer requests a service, the provider requests the black list of the requester. Using the black list records, a score is computed that determines the request's position in the providers upload queue. The provider can also chose to ask the requester to provide a limited list of favours it provided, in order to compute the score determining the upload queue position. It is assumed that all peers verify transaction receipts that are black listed in order to avoid malicious behaviour.

Summary: The concept of SeAl has the disadvantage that it requires that all peers are online all the time; otherwise transaction receipts cannot be verified. This verification is the core security mechanism, apart from signatures SeAl relies on. As the frequency of transaction record verification is unknown, it is hard to assess the traffic overhead created by SeAl. However, it is obvious that the verifications increase with the transaction frequency in the system. Furthermore, peers can easily collude and claim they provided each other services although they did not. This way they can influence their upload queue position at other peers. Finally, SeAl does not provide an objective view about the globally provided services of a peer, as each peer stores the transaction records locally and can choose which receipts it presents for verification.

Counteracting Free-Riding in Unstructured P2P Overlay Networks

In (KIKOU08), a system is presented by Karakaya et al. that aims at reducing free-riding in unstructured systems. This is especially interesting, as the systems presented so far are either storing information locally and are therefore independent of the used overlay network structure (see KaZaA (Sha04) and eMule (eMu04a)), or use a structured overlay to store accounting information remotely.



(a) Concept of Karma (b) Bucking freerider concept with mediator

Figure 2.4: Concept of "Karma" and "Bucking Free-riders" (cf. (VCS03, ABOS03),

Karagaya et al. use monitoring of a peer's neighbours in order to collect relevant information. Each peer counts messages of five different types it receives or forwards to the monitored node. Based on the ratio of the counted messages, it is judged if a peer is a free-rider or not. As a punishment of free-riders, their query messages are either ignored or reduced in scope, that is their time-to-live is reduced.

Summary: The system's design is very elegant as it does not introduce additional traffic for monitoring. However, it is also not able to record real accounting information and to provide it globally. Therefore, it is not useful to accounting in p2p systems.

2.3.2.2 Systems Using Receipts and Remote Accounts

Karma

A system taking into account a peer's actions in the overall system is Karma (VCS03). It is assumed that the peers are organised in a distributed hash table (DHT). Karma stores for every peer in the system a value that represents its balance of uploads against downloads. This balance is stored at remote peers. These remote peers are called a peer's bank set. The bank set consists of multiple peers, for redundancy reasons. The bank-set of a node A is located at the leaf-set of the node closest in the nodeID-space to $HASH(nodeId(A))$. A peer's balance must not be lost. Accordingly, a bank set is rather large - a suggested size is 64 peers. In a transaction where a peer A requests a service from peer B, first A informs its back set about the planned transaction. Then, the bank sets of the provider and receiver peer communicate in a any-to-any fashion in order to adjust the peers' balance according to the transaction value. Now B's bank set informs B that it can provide the service (see Figure 2.4 a)). Karma puts the focus on secure transactions. The maintenance of the bank sets is not explained. Further, Karma includes the concept of an epoch. At the end of each epoch all peers exchange messages about their account balance. Using this information the account balance is multiplied with a factor that prevents inflation. This requires $O(N^2)$ messages.

Summary: Karma is a fully decentralised accounting mechanism for p2p systems applying receipts and local accounts. Further, it provides a global view on the actions of peers. Therefore, it is the first systems that fulfils the core requirements of a p2p accounting mechanism.

However, Karma has three major drawbacks. The bank set of peers is known in the system. Therefore, it can be attacked by an adversary. Further, a peer could try to collude with the majority of its bank peers in order to change its account balance. The second drawback is the requirement that the bank set is always available. If all peers in a bank set go offline, the corresponding account owner cannot perform any transactions. There are two extensions of Karma, that resolve this; they will be discussed later in the respective section. The third disadvantage is the high transaction costs due to the complex transaction protocol.

Distributed Rating

In (DGGZ03), the concept of a distributed rating scheme is presented with the goal to prevent free-riding. Each peer in the network has k supervisors. In the ring-topology of a distributed hash table (DHT) each peer supervises its k immediate successors. There is one main supervisor and $k-1$ backup supervisors. After a transaction, the consumer peer rates the provider peer and sends this rating to the provider's main supervisor. The backup supervisors just keep a backup of the records, which they update periodically. When Chord is employed, a peer never has to route messages with its supervisor as destination, because its supervisors are located backwards in the Chord ring. Therefore, it is hard for a peer to manipulate its own rating; this would be discovered by the peers that route the message. In addition, a lightweight verification scheme is employed to ensure the correctness of peer ratings. When a peer requests a service, it sends with the request some of its accounting records where it provided a service. The potential service provider can now check the received records by querying the contained service receivers, and ask if these records are correct.

Summary: The distributed rating scheme stores receipts locally and remotely at supervisor peers. There are no further security mechanisms that ensure the trustworthiness of these receipts. The trustworthiness is achieved by the lightweight verification scheme. However, a peer can select which receipts it presents for checking. Also the concept for selecting the supervisors enables a peer easily to contact them directly. Overall, the scheme is not well elaborated and not useful for trusted storage of accounting records.

Bucking Free-riders

In (ABOS03), another system using remote accounts is presented. It is assumed that the peers are organised in a distributed hash table (DHT), and accountants store a peer's balance. Like in Karma, the accountants are a set of third peers. Peer accountants are determined using well known identifiers, which are computed using a hash function. Each peer can look up the accountants of all other peers. The set of accountants is maintained by each accountant doing periodical lookups for all accountant identifiers.

With every transaction, the balance of the two transaction partners is updated to the new value. Consistency among the accountants is achieved using the Small Byzantine Quorum (SBQ) protocol (MAD02). A non-mediated and a mediated settlement protocol are presented, where the mediated protocol can deal with some protocol violating behaviour. The mediated transaction protocol is depicted in Figure 2.4 b). The mediator is selected randomly. The mediator checks the buyers account balance and verifies it to the seller. The seller will provide the selected server encrypted with a randomly selected key, along with the key encrypted by the mediator. The buyer sends the encrypted key to the mediator as a means of notification that he received the service. Now the mediator initiates the transfer of the payment between the accounts. Each accountant of the seller confirms the receipt of the payment to the mediator. The transaction is completed when the mediator sends the first encryption key to the buyer, so he can decrypt the service.

Summary: Figure 2.4 b) shows that this model requires heavy communication for one transaction. Furthermore, the trust issue is projected to a mediator, which could potentially be controlled by one

of the parties. Because the accountants are publicly known, they could be also easily attacked by an adversary. The SBQ protocol can maintain account consistency if not more than $\frac{k-1}{3}$ accountants are subverted, where k is the number of accountants.

PeerMart

In (Hau05) Hausheer researched PeerMart, which is a system establishing a decentralised market place consisting of a decentralised auctioning mechanism (see also (HS05b)) and a decentralised accounting mechanism "PeerMint" (see also (HS05a)). PeerMint stores the peers' account in a distributed hash table. All messages sent are signed by the sender. Therefore, it belongs to the class of remote accounts with signed receipts. Also, PeerMint uses so-called session mediation peers to maintain session information about transactions. Although the session mediation peers are not trusted, it is assumed that they do not have an incentive to act dishonestly. PeerMint uses two kind of accounts. Session accounts hold accounting information about one specific transaction. In a transaction the session mediator holds the session account. Peer accounts hold aggregated accounting data of a peer, e.g., the amount of uploaded and downloaded traffic. Peer accounts are stored at remote peers. Tariffs bridge the gap between accounting events and balance updates. They are used to evaluate accounting events during a transaction (session). At the end of a transaction, a balance update is created by the session mediator and is then forwarded to the remote peers holding the transaction partner's accounts. This mechanism is thought to avoid a situation where the providers account receives an update but the consumers account is not debited. Also, if the session mediator observes a difference between the accounting events sent by provider and consumer peer to it, the transaction can be terminated immediately. For increased robustness, in each transaction m session mediator peers are used and all accounting messages are sent by both the consumer and the provider to all of them. Peer accounts are also replicated. Consistency between the peer accounts is achieved using majority decision. The location of the peer accounts is determined by the hash value of a peer's peer ID. The peer responsible for this value in the DHT is the so called root peer for this peer account. Its leaf-set is used to hold the peer account. Session accounts are identified using the hash value of the transaction's session ID. The responsible session mediation peers are again the session account's root peer's leaf-set. Messages sent to an account are first sent to the account's root peer, which forwards the message to the account holders. Maintenance of peer accounts is handled together with the maintenance of the DHT, i.e., when the leaf set changes, the peer account locations are updated. PeerMart uses Pastry (RD01) in its prototype implementation.

Summary: In conclusion, PeerMint is a fully distributed accounting mechanism for p2p systems. However, there are characteristics of PeerMint that create drawbacks. During a transaction, there is out-of-band traffic generated with several involved peers, the session mediation peers. This traffic scales linearly with the number of mediation peers m used. At the end of a session a the peer accounts have to be updated which has a message complexity of $O(mp)$, where p is the number peers holding a peer account. There are also trust issues with PeerMint: The account holding peers for a specific peer can easily determined by an adversary. An adversary needs to control only half of an account's account holding peers in order to manipulate the account. This enables also potential collision between one transaction partner and the mediation peers. A special weakness is the result of the root peers. If the root peers are malicious or controlled by an adversary, the peers can avoid that messages are forwarded to the account holders. A concrete scenario for an attack is that a peer A controls the root peer for its account, so in a transaction, peer A can avoid having its account debited for services rendered.

2.3.2.3 Token-based Systems with Remote Information Storage

Offline Karma

Offline Karma presented in (GH05) extends Karma (VCS03) such that it can operate even when bank sets are offline. In order to achieve this, electronic coins are introduced. Each peer can mint coins

by computing hash collisions. A coin consists of the minter's ID, a serial number, the issuing date, and the minter's signature. Each peer possesses a private/public key pair. Peers spend coins by adding the receivers ID to it, signing it and sending it to the receiver. The receiver can re-spend tokens as well, causing the size of the coin to grow over time. In order to detect double spending and reduce a coins size, it has to be reminted. That is, there is maximum amount of time a coin can exist, and it has to be reminted at least once in its lifetime.

Reminting is performed by a set of peers. The set is composed such that at least one peer is a non-corrupted node and all honest reminters possess the reminted coin's history of previous remints. The set of peers responsible for reminting a specific coin is randomly selected and predetermined. That is, for a coin the responsible reminter set are the closest neighbours of the hash value of the coins identifier. This ensures that the reminter set is not manipulated by measuring the density of peers in the DHT. The reminter set peers must not be outside a specific range within the DHT. The maintenance of reminter sets is not further elaborated on.

For reminting, the coin is sent to each peer in the reminter set that verify the correctness of the coin. Then the reminters create a multisignature for the new coin with the same coin identifier and owner, but with a new time stamp. If verification fails, the dishonest peer is identified and blacklisted.

Summary: Offline Karma presents an interesting concept for a decentralised virtual currency scheme. Attacking the scheme at the reminting requires a lot of effort for an adversary with little gain. A peer planning to defraud the system has to control a different set of peers for each coin it wants to gain. Not explained is the permanent storage of coin history in order to detect double spending. However, this is a very important functionality.

Overall the scheme does not present a solution for accounting. With Offline Karma transaction receipts for a specific peer cannot be determined. transaction records could be only collected on a per coin basis. Modifying the scheme to store information on a per peer basis would raise further security concerns. Then an adversary could gain much more from attacking a reminter set. Therefore, changing this characteristic would require additional security mechanisms.

In summary, although an interesting and probably efficient approach for a virtual currency, Offline Karma cannot be applied for accounting.

Running on Karma

Running on Karma (Cho07) is a modification of Offline-Karma (GH05) and presents a reputation mechanism as well as a virtual currency system. Here, the virtual currency system is discussed, as it is closest to an accounting functionality.

Minting of coins happens like in Offline Karma. Also, like in Offline Karma for each coin a neighbour set of peers exists that store information about that coin, especially its current owner. Accordingly, the neighbour set is the coin's "bank-set". It differs from Offline Karma, in that here a coin does not expire. The first time a coin is used, it is sent to its bank-set in order to register it. Then the new owner and a timestamp are added to the coin, the current owner signs the coin, and then sends it to the new owner. Similarly, a coin can be re-spent by a user. The current user informs the coin's bank-set about the new user. When the new user receives the coin it will also check with the coin's bank to verify that the sender was the previous owner. This way, double spending can be detected and prevented.

Summary: Running on Karma presents an efficient approach for a virtual currency for p2p system. It is especially interesting because it does not require reminting like in Offline Karma; therefore Running on Karma does not require complex security mechanisms. This is enabled by storing the coins ownership in the bank-sets. In contrast to Offline Karma, these bank-sets have to be available all the time and have to reliably store the coins owner. The maintenance mechanisms for this are not elaborated.

For using Running on Karma as an accounting scheme the same constraints apply as for Offline Karma. Unfortunately, it cannot be employed for this purpose.

2.3.2.4 Token-Based Systems with Central Information Storage

Into this category also some micropayment schemes fall. For p2p systems MojoNation (that was renamed MNet after the MojoNation stopped to exist) is an example, where tokens were used as an incentive system.

MojoNation/MNet

MojoNation (Wik08b, Moj00) was one of the earliest Peer-to-Peer systems to use a payment protocol. Users had to use a virtual currency called Mojo to obtain a service from another peer. MojoNation still required a centralised trusted third party to issue the Mojo and to resolve double-spending issues.

Every action in MojoNation had some price in Mojo; this was thought to prevent denial-of-service attacks. When a consumer requested a file from provider, the consumer sends an "I-Owe-You" message to the provider. The provider will give the consumer credit until the credit is worth at least one Mojo; then, the Mojo is transferred from the consumers account to the providers account at the central token server. This concept reduces the load of the central server. Due to the "I-Owe-You"-concept trading is possible even if the central server is offline.

After MojoNation was discontinued, the open source project MNet (mne05) continued MojoNation's basic ideas. The MNet project was discontinued in August 2005.

Summary: Overall, the MojoNation concept received a lot of attention when it was released, because it was a completely new concept for file sharing. However, it missed a concept for decentralising the issuing and administration of Mojos. Therefore, it is not applicable to p2p systems.

2.3.2.5 Token-Based Systems with Local Information Storage

Trading In Stamps

A system using stamps for peers' "evidence of participation" is presented in (MT03). Every peer issues personalised stamps and trades them with other peers. If peer A requests a service from peer B, peer A has to pay a specific number of peer B's stamps back to B. In such a community, a market for stamps should develop. There is no limit how many stamps a peer issues. The author envisions that if a peer issues too many stamps in comparison to its offered services, their stamps will devalue due to market rules. Thus, the peer will have difficulty obtaining other stamps, as rational nodes will not wish to purchase its stamps. This way the stamps protocol shall combine a virtual currency and reputation.

Summary: It is questionable if this concept is viable, because for a market to develop there are some prerequisites missing. For example, there is no information on the quality of the the services traded. A user would also have to establish an exchange rate for each users stamps he encounters. Without sufficient information, which the system does not provide, a user cannot estimate a stamp's value. More important, there is no global accounting information available, because the stamps are traded.

PPay

As mentioned before, micropayment systems use a central broker or bank in order to host accounts or to issue virtual currency. Thus, they are not appropriate for Peer-to-Peer systems. A micropayments system tailored to Peer-to-Peer systems is presented in (YGM03). It relieves the broker of some tasks and gives them to the peers to be facilitated. As a result the broker can go off-line for short time periods and the system can still continue to operate. PPay uses electronic coins that are issued by a central broker, to a specific peer. In order to pay for a service, a peer adds the receiver and a sequence number to the coin and signs it with its own private key. The receiver has the alternative to either cash in the coin at the central broker or to re-spend the coin with another peer. In order to do that, it contacts the coin's owner and requests a reassignment. That is, the owner replaces information on the old receiver with information on the new receiver, increases the sequence number, and then sends this coin to both

receivers. Therefore, the coin does not grow in size when it is reassigned[4]. The central broker charges a small flat fee per caching request, regardless of the number of coins being cached, giving peers an incentive to reassign received coins instead of caching them. This releases the broker of load.

Like other micropayment schemes PPay is subject to fraud; peers cannot forge coins, because they do not have knowledge of the broker's private key. However, peers can double spend coins. Double spending control in PPay is performed by the broker or the coin's owner. When the coin's owner double spends a token, the broker will realise it when the coin is cashed the second time. When a coin holder reassigns a coin more than once the coin's owner will recognise this, as he has to sign the reassignment. Further, a coin owner cannot claim a reassignment is invalid, as he signs each reassignment and increases the serial number.

Summary: Overall, PPay is a very elegant system that releases the central broker of load by allowing electronic coins to be respent in a p2p system. The design even moves part of the security mechanisms for fraud detection to the peers allowing the system to operate if the broker is offline. However, a central broker is still required.

2.3.3 Summary

There exist different mechanisms in p2p systems that provide accounting functionality. A classification for these mechanisms is presented that distinguishes the mechanisms according to the type of accounting information stored, the location where the information is stored, and (for mechanisms that use pre-issued documents (tokens)) how these tokens are issued. In Figure 2.3 the existing p2p accounting mechanisms are classified accordingly.

The discussion of the existing systems shows that some of those mechanisms cannot be applied to p2p systems, either because they rely on a central trusted entity (e.g., accounting in grids, CDNs, RADIUS, micropayments) or because they are not suited to provide accounting information (e.g., Proof of Work, micropayments). The remaining p2p accounting systems can be classified into four different types of mechanisms (see Table 2.1).

The first type (receipt based, local accounts) is well known from existing file sharing applications. At the peer, a balance of upload traffic and download traffic is stored locally. In eMule (eMu04a, eMu04b) the information is used only to manage the local upload queue. Therefore, this system does provide a local view about the provided resources and services. Here, a peer does not have incentives to cheat. In KaZaA (Sha04) global accounting information is stored in local accounts. This can be bypassed by using a manipulated client software (Wik08a). This shows, that using local accounts global accounting information cannot be stored in a trusted manner. Accordingly, this design alternative is not suited for trusted accounting in p2p systems.

The second type uses remote accounts in order to store global accounting data per peer. A bank set can offer accounting information about all transactions a peer performed system-wide. In order to deal with churn of peers, these accounts are replicated over a set of peers. This requires further mechanisms to maintain the sets, which are seldom explained. During transactions, these systems require messages between the transaction partners and their account holding peers. Often, this traffic has a message complex of $O(k^2)$, where k is the size of the set of peers holding an account. In this class of accounting systems, Karma (VCS03) and PeerMart (Hau05) offer the characteristics that are closest to the goal of this dissertation. Both mechanisms also offers efficient cooperation control. However, these systems include trust issues, because they only rely on the remote accounts. Storing the accounts at third party peers does not provide trustworthiness, if these peers can be attacked or these peers can collude. Hence, a trustworthy accounting mechanism cannot rely on remote accounts alone. Additional mechanisms are required. Further, during transactions a message complexity of the traffic overhead lower than $O(k^2)$ is desirable.

[4] Typically in coin based micropayment system the coin grows in size because the new receiver is simply added to the coin.

Table 2.1: Overview Existing P2P Accounting Mechanisms

	Local Account	Remote Accounts
Receipt based	• eMule (eMu04a) • incentive compatible p2p (NNS+04) • KaZaA (Sha04) • SeAl (NT04) • Swift (TPM04)	• Bucking Freeriders (ABOS03) • Distributed Rating (DGGZ03) • Karma (VCS03) • PeerMart (Hau05)
Token based	• Stamps (MT03)	• Offline Karma (GH05) • Running on Karma (Cho07)

The third type of accounting mechanisms uses self issued documents (tokens) which stored in local accounts. An example system is Trading in Stamps (MT03). Here, peers trade their tokens against services. Thus, message complexity during transaction is $O(1)$. Because tokens are traded, accounting information is not stored. It is only an incentive system. Also its effectiveness is questionable. Furthermore, there is no effective cooperation control possible, because peers can issue an unlimited number of tokens.

The forth type of accounting mechanism also uses self issued tokens, but stores these in remote accounts. These mechanisms are "Offline Karma" (GH05) and "Running on Karma" (Cho07). In these systems each remote account is responsible for only one token. Therefore, attacking a remote account or collusion has a very limited gain for the adversary. Thus, attacks do not pay off. Message overhead during a transaction has a complexity of $O(k)$. However, both systems do not provide the functionality to query accounting information about a specific peer. Therefore, they are not applicable to accounting.

In sum, Karma (VCS03) and PeerMart (Hau05) are the two accounting mechanisms which are closest to the requirements for trusted accounting in p2p systems with intrinsic cooperation control. Unfortunately, both mechanisms still raise trust issues.

In order to design an accounting mechanism that fulfils all requirements, a gap in the solution space could be identified. No system currently exists that combines local storage of accounting information with decentralised issuing of tokens. However, this promises to be the most efficient solution for a p2p accounting mechanism, as combines a message complexity of $O(1)$ during transactions with the p2p approach. Also, transactions could be performed in such a system, even if specific third peers are offline or specific accounting information, e.g., bank account information, is currently not available due to churn. In the next section security mechanisms are discussed that allow decentralising a broker.

2.4 Distributed Security Mechanisms

A main objective of this dissertation is to build a trustworthy accounting scheme for p2p systems without usage of a trusted entity in the system, so a distributed basis of trust required. This section introduces

security mechanisms that can be applied to autonomous distributed systems, as they do not rely on a centralised trusted entity.

2.4.1 Multisignatures vs. Threshold Cryptography

There are two different types of security mechanisms that allow for the creation of a document that can be signed by several signees, that is, a central trusted entity is not a requirement. These are multisignature schemes and threshold cryptography schemes that build on secret sharing.

In multisignature schemes (Oka88, MOR01, Bol03) multiple peers sign a message sequentially; the s_i is computed using the s_{i-1} signature and the signees private key. The resulting signature has the length of a single signature. To verify the signature all public keys of the signees are required. Many multisignature scheme support a verification independent of signature sequence. However, for verification the signee's public key must be known. As signees can be offline when a verification is required, multisignature schemes require the availability of the certification authority that issued the peers' key pair. This would heavily rely on the use of a central trusted entity in order to establish trust in p2p systems. Therefore, multisignature schemes are not a valid solution for building a fully decentralised trustworthy accounting scheme for p2p systems.

The second alternative for creating signatures using several signees is threshold cryptography. Threshold schemes use secret sharing to split a key into many parts, called shares. A specific threshold of signees is required to create a signature with the key. Each signee creates a partial signature with its key share and all signatures are then combined by the peer requesting the signature. The created signature is anonymous. The signature verification is performed with the corresponding public key to the shared private key. Furthermore, key pairs can be generated and managed in a completely decentralised way using proactive secret sharing.

In conclusion, threshold cryptography can be applied in p2p systems without relying on a certification authority, as multisignature schemes require. The process reviewed below includes how to share a secret, i.e., a secret private key, among the participants of the distributed autonomous system (or among a group of its participants) and how to apply the shared secret in order to sign documents in a distributed, trusted manner. It is of utmost importance that the secret is never compromised nor known to a single participant. Otherwise, it would be necessary for the participant to be trusted, which we cannot assume.

This section starts with the introduction of secret sharing and extends the concept to threshold cryptography and proactive secret sharing.

2.4.2 Secret Sharing

A main objective of this dissertation is to build a trustworthy accounting scheme for p2p systems without usage of a trusted entity in the system, so a distributed basis of trust required. This section introduces security mechanisms that can be applied to autonomous distributed systems, as they do not rely on a centralised trusted entity.

2.4.3 Multisignatures vs. Threshold Cryptography

There are two different types of security mechanisms that allow for the creation of a document that can be signed by several signees. These are multisignature schemes and threshold cryptography schemes that build on secret sharing.

In multisignature schemes (Oka88, MOR01, Bol03) multiple peers sign a message sequentially; the s_i is computed using the s_{i-1} signature and the signees private key. The resulting signature has the length of a single signature. To verify the signature all public keys of the signees are required. Many multisignature

scheme support a verification independent of signature sequence. However, for verification the signee's public key must be known. As signees can be offline when a verification is required, multisignature schemes require the availability of the certification authority that issued the peers' key pair. This would heavily rely on the use of a central trusted entity in order to establish trust in p2p systems. Therefore, multisignature schemes are not a valid solution for building a fully decentralised trustworthy accounting scheme for p2p systems.

The second alternative for creating signatures using several signees is threshold cryptography. Threshold schemes use secret sharing to split a key into many parts, called shares. A specific threshold of signees is required to create a signature with the key. Each signee creates a partial signature with its key share and all signatures are then combined by the peer requesting the signature. The created signature is anonymous. The signature verification is performed with the corresponding public key to the shared private key. Furthermore, key pairs can be generated and managed in a completely decentralised way using proactive secret sharing.

Summary

In conclusion, threshold cryptography can be applied in p2p systems without relying on a certificate authority, as multisignature schemes require. The process reviewed below includes how to share a secret, i.e., a secret private key, among the participants of the distributed autonomous system (or among a group of its participants) and how to apply the shared secret in order to sign documents in a distributed, trusted manner. It is of utmost importance that the secret never be compromised nor known to a single participant. Otherwise, it would be necessary for the participant to be trusted, which we cannot assume.

This section starts with the introduction of secret sharing and extends the concept to threshold cryptography and proactive secret sharing.

2.4.4 Secret Sharing

This section presents basic secret sharing schemes and evaluates their usefulness for application in autonomous distributed systems like p2p.

Secret Sharing can be defined as follows: A trusted dealer TP gives each player P_i a secret share ss_i of the secret s in such a way, that any group of t or more players can together reconstruct the secret, but no group fewer then t players can do this. Here, t is denoted the threshold and n is the total number of players.

A simple secret sharing technique is so-called XOR Secret Sharing. Here, the secret s is split into $n-1$ random parts and part s_n is the result of combining all the parts $s_1, s_2, \ldots, s_{n-1}$ using the XOR-function. It is not applicable for signing messages, but can be used to explain the basic concept of secret sharing. Further details are given in Appendix C.2.1.1.

Another often used secret sharing technique is called "*additive secret sharing*". This secret sharing scheme is often used in threshold cryptography scheme. In additive secret a secret s is split into n parts, and part s_n is computed from the other $n-1$ parts. When a secret key is created and shared among peers, all shares have to be created at the same time. Additional shares cannot be created later. Therefore, additional secret sharing is not applicable to p2p system, where membership is dynamic. Additive secret sharing is used and presented in (Rab98) and (JS05). Further details about additive secret sharing are given in Appendix C.2.1.2.

Another secret sharing scheme was developed by Blakely et al. in (Bla79). Its major disadvantage is that the secret has to be completely reconstructed in order to create signatures. In p2p systems a peer would then learn the secret key, which is not acceptable. Further details are given in Appendix C.2.1.3.

2.4.4.1 Shamir's Secret Sharing

Shamir's secret sharing scheme is one of the most important secret sharing schemes. It is presented in (Sha79). Many other secret sharing schemes are based on it. The goal of (Sha79) is to divide a secret s into n pieces s_1, ..., s_n(called shares) in such a way that:

1. knowledge of any k or more s_i pieces makes s easily to computable;
2. knowledge of any $k-1$or fewer pieces leaves s completely undetermined (in the sense that all its possible values are equally likely). (cf. (Sha79))

In the literature, such a scheme is called (k, n)-threshold scheme. This means that one needs at least k shares to reconstruct the secret and that it is not possible to reconstruct the secret with at most $k-1$ shares. To obtain a robust threshold scheme, Shamir proposed creating at least $n = 2k-1$ shares. If $k-1$ shares are corrupted, it is guaranteed that at least k shares are not corrupted and the secret can be reconstructed correctly.

For reconstruction polynomial interpolation is used. With polynomial interpolation it is possible to reconstruct a polynomial $f(x)$ of degree $k-1$ with k points of the function. In general, a dealer creates a function $f(x)$ that represents the secret s at $f(0)$ ($f(0) = s$) and distributes points of the function to the different shareholders. k of this points are then used for reconstruction of the secret s. The details are described in the next section, and follow the work of Shamir (Sha79).

Initialisation Phase

In the initialisation phase, a dealer generates the shares of the secret s and transmits these shares to the different shareholders. The following steps are performed:

1. The dealer D creates a function $f(x)$ of degree $k-1$ with $f(0) = s$ (using Equation 2.1)

$$f(x) = s + \alpha_1 x + \alpha_2 x^2 + ... + \alpha_{k-1} x^{k-1} \bmod q = s + \sum_{i=1}^{k-1} \bmod q \tag{2.1}$$

whereas $\alpha_1, ..., \alpha_{k-1}$ are random numbers modulo a prime q. The prime q has to greater than the value of the secret s and the maximum index (x_i) used to generate a share.

1. D computes $n = 2k-1$ points (x_i, y_i);

$$\{y_i = f(x_i) \mid i \in \{1, \ldots, n\} \wedge x_i \neq x_j \ for \ i \neq j\} \tag{2.2}$$

   These points represent the shares ($s_1 \ldots s_n$) of the secret s.

2. These shares $(s_1, \ldots, s_n)$ are then distributed secretly to the corresponding shareholders.

Reconstruction Phase

To reconstruct the secret, at least k shareholders have to cooperate. Like mentioned above, reconstruction is based on polynomial interpolation. The Lagrange interpolation is presented in Equation 2.3 (from (BAM00)). y_j represents the shares of the shareholder P_j and x_j is value of the variable used to generate y_j. In the following x_j is called index, because it is assumed that $x_j = j$ for shareholder P_j.

$$f(x) = \sum_{j=1}^{k} y_j \prod_{i=1, i \neq j}^{k} \frac{x - x_i}{x_j - x_i} \bmod q \tag{2.3}$$

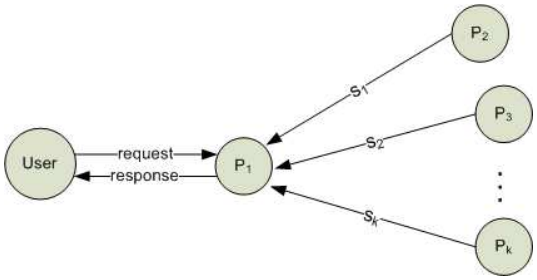


Figure 2.5: Classical reconstruction of a secret

A simple approach to reconstruct the secret, is to send at least k shares to a shareholder that reconstruct the secret using Equation 2.3 (see figure 2.5).

Since the complete reconstruction of the secret s offers the possibility to an attacker to get s other approaches were developed. Some approaches to use the shares for a distributed generation of a RSA signature are presented in Section 2.4.7.1.

2.4.4.2 Summary

In order to apply secret sharing within the autonomous distributed systems space (more specifically p2p) a secret sharing scheme should fulfil a set of requirements.

In autonomous distributed systems nodes join and leave the system in an unpredictable manner. So, it can not be assumed that all nodes will be online at a specific time. Therefore, (n, n)-secret sharing schemes can not be applied, but only (k, n)-secret sharing schemes. Thus, XOR-sharing is not suited for autonomous distributed systems.

Further, new nodes might join the system after the initial sharing of the secret and such nodes should also receive a share. Accordingly, when in the initialisation phase a specific share should not be calculated based on all other shares created. This would mean that no new shares could be created after the initialisation phase. Therefore, additive sharing is also not suited for autonomous distributed systems.

As a last requirement, it is important that no single node can learn the secret when the shares are used either to sign a message or to encrypt a message. Shamir's secret sharing offers this ability, however, Blakely's scheme doesn't. This ability is explained in further detail in the following section.

In conclusion, the secret sharing discussion, there is only Shamir's secret sharing scheme that is suited for autonomous distributed systems. Therefore, in the following, only security mechanisms applying Shamir's secret sharing are discussed.

2.4.5 Verifiable Secret Sharing

In order to increase the security of the classical secret sharing schemes, verifiable secret sharing (VSS) schemes were developed. These schemes offer the shareholders the possibility of verifying the shares that are distributed by the dealer. The schemes presented above do not offer such possibilities and thus, a malicious dealer can destroy the secret s by distributing malicious shares.

Only Pedersen's Verifiable Secret Sharing Scheme is presented here because it is employed in the threshold cryptography scheme selected later for usage in p2p systems. A comparison and evaluation of different Verifiable Secret Sharing schemes is given in Appendix C.2.2.

2.4.5.1 Pedersen's Verifiable Secret Sharing Scheme

Petersen's scheme presented in (Ped91b) is based on Sharmir's secret sharing scheme and uses a slightly different approach than Feldman et al. It differs from Feldman et al. in that it is based on two constants g, h.

Initialisation Phase

The dealer chooses two prime number p, q, with $p = mq + 1$ (m is a small integer), and chooses two $g, h \in \mathbb{Z}_p$ of order q. These constants are known by all participants of the verifiable secret sharing scheme. It is assumed, that $\log_g(h)$ is not known. Then the dealer D performs the following steps:

- D generates two random polynomials $f(x)$, $g(x)$ of degree $k-1$, so that $f(0) = s$ and $g(0) = random$ (see Equation 2.1) and computes the k verification values F_i for the coefficients $(f_i, g_i | i \in \{1 \dots k\})$ of these polynomials $f(x)$, $g(x)$ using Equation 2.4. These verification values are then broadcast to the shareholders.

$$F_i = g^{f_i} * h^{g_i}; i \in \{1 \dots k\} \tag{2.4}$$

- Then the shares are computed using $f(x)$ and $g(x)$ and Equation 2.2. The share tuple $(s_i, t_i) = (f(i), g(i))$, is transmitted secretly to the corresponding shareholder P_i,

Each shareholder then checks the correctness of the received share using Equation 2.5

$$g^{f_i} * h^{g_i} = \prod_{j=0}^{k-1} F_j^{i^j} = g^{\sum_{j=0}^{k-1} f_j * i^j} * h^{\sum_{j=0}^{k-1} g_j * i^j} \tag{2.5}$$

If the check of shareholder P_i fails, the shareholder broadcasts an accusation against the dealer and the dealer has to defend itself. Therefore, the dealer publishes the share sent to shareholder P_i. Now, the other shareholders have the task of determining whether the dealer D or P_i is malicious. If more then k accusations are broadcast, the dealer is malicious.

Reconstruction Phase

The reconstruction of the secret s is based on the shares s_i and is similar to the reconstruction phase presented in Section 2.4.4.1. At least k shares are sent to one shareholder that verifies the received shares using Equation 2.5. If the equation holds for at least k shares, these shares are used to reconstruct the secret. Otherwise an accusation is broadcast that has to be resolved by revealing the corresponding share. Additionally, it is possible to generate partial signatures.

Summary

Verifiable secret sharing adds an important property to secret sharing techniques. Verification values are published that each participant can use to check the validity of its share. Further, no information about the secret is revealed. However, Pedersen still assumes a central dealer for creating the secret.

2.4.6 Secret Sharing Without a Dealer

The previous Section showed how a secret can be shared by a dealer that can behave maliciously. However, in autonomous distributed systems we cannot allow one peer to have the knowledge of a system-wide secret key. Therefore, other methods will now be presented to generate and share a secret in a decentralised way such that no participant learns the secret.

2.4.6.1 Distributed Generation of Random Values

Pedersen presents in (Ped91a) Joint Secret Sharing, which enables Shamir's Secret Sharing without the need of a central dealer. The basic concept is to share the polynomial such that $f(x) = f_1(x) + ... + f_n(x)$, where $f_i(x)$ is the polynomial of participant P_i. The generating parties have to agree on the threshold t and further prerequisites, depending on the purpose of the random value to be generated. Then each participant P_i chooses a random polynomial of degree $t-1$such that $f_i(0) = s_i$, where s_i is the random secret that P_i selects. Each P_i computes $P_j^{(i)}$'s shares $\hat{s}_j^{(i)} = f_i(id_j)$ and sends it over a secure channel to P_j. Each P_i is doing this for all other participants P_j. Once P_j receives all shares $\hat{s}_j^{(i)}$ it computes its share $s_j = \sum_{i=1}^{n} \hat{s}_j^{(i)}$. During this process Verifiable Secret Sharing can be applied in order to check the validity of the shares s_j. Once each participant has its own share s_j, the secret can be recovered jointly using Lagrange interpolation as in Shamir's Secret Sharing.

A variant of Joint Secret Sharing where the secret is zero is called Joint Zero Secret Sharing (JZSS). It was presented in (HJKY95) and is use in Proactive Secret Sharing, which will be discussed later.

A variant of (Ped91a) is presented in (GJKR99, GJKR07). It resolves security holes where an attacker controls a small number of participants. It uses two broadcasting rounds during the key generation.

Summary

Distributed generation of random values is a well understood and secure technology. It extends Shamir's Secret Sharing in order to generate a random value (the secret) in a group of participants in such a way, that no participant learns the secret but all participants receive a valid share. Hence, distributed generation of random value is a base technology for applying distributed security mechanisms in systems without a trusted entity, such as p2p systems. Here, distributed value generation can be used, e.g., to generate a system-wide secret in a secure way. This can e.g. be applied to generate a distributed system-wide DSA key. RSA keys require specific characteristics of the random values the keys are based on. The next section presents an algorithm how to generate RSA keys in a fully distributed manner.

2.4.6.2 Distributed RSA-Key Generation

RSA is based on secret prime numbers. Therefore, the methods presented above cannot be applied to generate a secret RSA shared key in a decentralised manner. Boneh and Franklin presented such a mechanism in (BF01).This mechanism enables the generation of an RSA modulo $N = pq$ and a private/public key pair with exponents d and e, where $d \cdot e = 1 \bmod \varphi(N)$. At the end of the mechanism N and e are public and d is shared among the t participants.

The mechanism applies the scheme for distributed generation of a random number in order to generate two random numbers p and q. The challenge is that p and q should be prime numbers. In order to check this, another participant is selected testing whether the numbers are not divisible by a prime less than some bound B_1. If this fails, the random number generation is repeated. When the test succeeds, the t participants calculated $N = (p_1 + ... + p_t)(q_1 + ... + q_t)$. Now, using the so called biprimality test the participants have to test, if N is the product of two primes. If not, the protocol is restarted from the beginning. If so, the keys can be generated.

Summary

From this overview it becomes obvious that the mechanism has the disadvantage of generating distinct traffic between the participants, because due to the randomness it parts of the protocol have to be repeated many times. In (ABF+99) the results show that depending on the desired key length the required traffic grows exponentially. With 3 participants for a key length of 768 bits traffic of 1,02 MBytes, for a key length of 1024 bit traffic of 3,58 MBytes, and for a key length of 1536 traffic of 9,94

MBytes were generated per participant. Thus, compared to distributed DSA-key generation, distributed RSA-key generation is very expensive in term of traffic.

In (FMY98) a modification of the mechanism from Boneh and Franklin was presented, making the mechanism robust against a minority of adversaries among the Partial Signing Servers. It enables the identification of those parties who try to cheat during the key generation.

2.4.7 Threshold Cryptography

The approach to reconstructing the secret s is not secure, if it is not reconstructed at a trusted entity. Therefore, other approaches were developed. These approaches enable the generation of distributed signatures. This means that a coalition of shareholders generates partial signatures using their shares of a shared private key and send these partial signatures to the signature requester. The requester combines these partial signatures to get the final signature. Some approaches are presented in (FGMY97b, FGMY97a, JS05, Sho00, Rab98, GJKR01, HJJK97). A few of these ((FGMY97b, JS05, Rab98)) apply additive sharing. However, this would imply that all private key shares must be generated at system setup. This is not applicable to peer-to-peer systems, where no trusted central authority exists. Therefore, such schemes are not further considered.

Threshold cryptography is specific to the example cryptography scheme previously presented in this chapter. We now turn to a different scheme, the frequently discussed RSA-based threshold signature scheme.

2.4.7.1 RSA Threshold Cryptography Fundamentals

The assumption is that some shareholders share a private RSA key using Shamir's secret sharing scheme and have to sign a message m. Therefore, the following steps are performed.[5]

- The shareholders are requested to sign a message m, requiring a coalition of k shareholders communicate and distribute their indexes.
- Each shareholder P_i computes a partial key k_i using the summand of Equation 2.3 (see Equation 2.6).

$$k_i = s_i \prod_{j=1;j\neq i}^{k} \frac{-j}{i-j} \tag{2.6}$$

- With these partial keys k_i each shareholder computes a partial signature $sig_{s_i} = m^{k_i} \bmod N$ and sends this message secretly to the requester of the decryption task (see figure 2.6).
- The requester receives the k partial signatures $sig_{s_i}(m)$ and multiplies them to decrypt the message (see Equation 2.7).

$$\begin{aligned} sig(m) &= \prod_{i=1}^{k} sig_{s_i}(m) \bmod N = \prod_{i=1}^{k} m^{k_i} \bmod N = \prod_{i=1}^{k} m^{s_i \prod_{j=1;j\neq i}^{k} \frac{-x_j}{i-x_j}} \bmod N \\ &= m^{\sum_{i=1}^{k} s_i \prod_{j=1,i\neq j}^{k} \frac{-x_i}{i-x_i}} \bmod N = m^d \bmod N \end{aligned} \tag{2.7}$$

It is possible that the shareholders only compute the partial signed message m^{s_i}. These message are then sent together with the indexes i to the requester. The requester can then choose any k messages to

[5] Note that the scheme described below does not work for RSA.

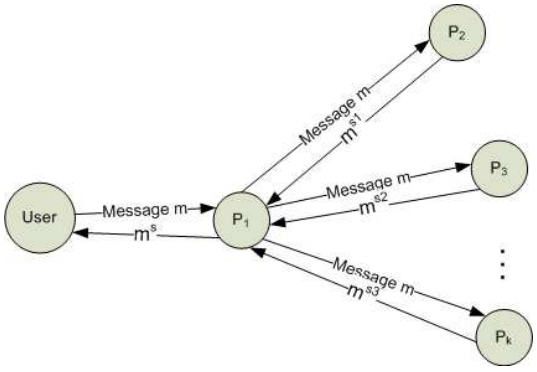


Figure 2.6: Threshold signature without revealing the secret

generate the final decrypted message. Just the product $\lambda_i = \prod_{j=1;j\neq i}^{k} \frac{-j}{i-j}$ of the corresponding partial message m^{s_i} has to be generated and the m^s can be created.

$$m^s = \prod_{i=1}^{k} (m^{s_i})^{\lambda_i} \bmod N \tag{2.8}$$

For many years, threshold cryptography seemed unable to be applied to RSA-based cryptography. This is due to the negative numbers or non-natural numbers in the exponent of the RSA-threshold signature that are created by the Lagrange interpolation (e.g. see (DF89)).

If the Lagrange coefficient is negative, one has to compute the inverse modulo of m. This can be done efficiently, if the message m and the RSA modulo N are relative prime (see (Buc03)). However, to compute the exponentiation of a message $m^{\frac{a}{b}} \bmod N$, one has to compute the root modulo N. The only way to compute this root is to know $\varphi(N) = (p-1)(q-1)$ (see (Buc03)). This requires knowledge of the primes p and q that must remain secret. Therefore, one has to avoid fractions in the exponent. Several possibilities to accomplish this exist ((FGMY97a, Sho00, LKZ+04)). However, these options generate additional problems. Either they apply additive secret sharing, or they use a different method, which was first presented by Frankel et. al in (FGMY97a), presented below.

2.4.7.2 Frankel's RSA-based Threshold Cryptography Scheme (Summary)

The scheme of (FGMY97a) can be applied to both polynomial secret sharing (Section 2.4.4.1) and additive secret sharing (Section C.2.1.2). The principle is to avoid the non-natural numbers of the Lagrange coefficients by extending the exponent by the factorial $n!$, where n is the total number of shares issued. This way, non-natural numbers in the exponent can be reduced. Details about Frankel's threshold scheme are given in Appendix C.2.2.5.

Summary

Although this scheme circumvents the problem of negative numbers or non-natural numbers in the exponent, it has a limitation that prohibits its application large distributed anonymous systems. In order to avoid non natural numbers in the Lagrange coefficients, one has to compute the factorial of the number of shareholders. This is feasible for threshold groups with around up to 100 shareholders, but not for threshold groups with thousands of shareholders. Accordingly, this scheme can be applied in server-based environments like COCA (ZSvR02). However, it is not suited for large p2p systems.

2.4.7.3 Further RSA-based Threshold Signature Schemes

Like in (FGMY97a), the same problem exists in the RSA-based threshold cryptography scheme presented in (Sho00). This scheme also enables the verification of the shares.

The only scheme that seems applicable to large autonomous distributed systems was presented by Luo and Kong in (KZL+01, LKZ+04). This scheme will be called "URSA" in the following.

2.4.7.4 USRA: RSA-based Threshold Cryptography Scheme

A protocol to provide ubiquitous and robust access control (URSA) in mobile ad hoc networks without a centralised authority is presented in (LKZ+04).

Initialisation Phase

The dealer D generates the RSA parameters (RSA modulo N, public key e, private key d) and creates a random polynomial $f(x)$ of degree $k-1$ with $f(0) = d$. Then the shares $s_i = f(i) \bmod N$ are computed. In general this phase is similar to Shamir's secret sharing (Section 2.4.4.1) except that the shares are reduced modulo in the RSA modulo.

Reconstruction Phase

In contrast to the RSA sharing schemes presented in Section 2.4.7.1, this scheme is not limited to a small number of shareholders. This is achieved by introducing the "k-bounded offsetting algorithm" to reconstruct the secret.

To create a signature, k shareholders have to cooperate and generate the partial signatures using Equation 2.9 and then send these signatures to the signature requester.

$$sig_{s_i}(m) = m^{s_i \prod_{j=1;j\neq i}^{k} \frac{-j}{i-j} \bmod N} \bmod N \tag{2.9}$$

To reconstruct the final signature the requester multiplies the received signatures. However, Equation 2.10 shows, the resulting signature is not equal to the final signature, because the summation of the exponents is not reduced modulo N.

$$\begin{aligned} sig_s(m) &= \prod_{i=1}^{k} sig_{s_i}(m) \bmod N \\ &= m^{\sum_{i=1}^{k}(s_i \prod_{j=1;j\neq i}^{k} \frac{-j}{i-j} \bmod N)} \bmod N \\ &= m^{d+tN} \bmod N \neq m^d \bmod N \end{aligned} \tag{2.10}$$

To eliminate the part m^{tN} the range of values of $s_i \prod_{j=1;j\neq i}^{k} \frac{-j}{i-j} \bmod N$ has to be viewed. These values are in the interval $[0, \ldots, n-1]$ and thus, the value of parameter t is in the set $\{0, 1, \ldots, k\}$. To reconstruct the final message, the requester multiplies m^{d+tN} with m^{-N} as long as the resulting message can be decrypted with the public key e to the original message m (see Equation 2.11). At most k attempts are needed ($\alpha = \{0, 1, \ldots, k\}$), otherwise a partial signature is false.

$$m \equiv (m^{tN+d} m^{-\alpha N})^e \bmod N \tag{2.11}$$

Summary

URSA circumvents the issues of other RSA-based threshold cryptography schemes by introducing the "k-bounded offsetting algorithm". Thus, this is the first threshold cryptography scheme which is actually applicable in large p2p systems. Still, for RSA keys the distributed generation of a new key is very expensive in terms of traffic. Thus, next DSA-based threshold cryptography scheme are discussed, as here distributed key generation is much more efficient (see above).

2.4.7.5 DSA Threshold Cryptography

Basics

The Digital Signature Algorithm (DSA), also known as Digital Signature Standard (DSS), is based on the El Gamal signature scheme (Gam85) and was approved in 1994 by the National Institute of Standards and Technology (NIST) (Nat94).

DSA uses three system-wide parameters p, q, g. Parameter q is a 160-bit prime number, p is a large prime number such that q divides $(p-1)$, and g in an element of order q in $\mathbb{Z}_p^*$. As the private key, users select a random integer $x \in \mathbb{Z}_q$. The corresponding public key computes to $y = g^x (\mathrm{mod}\, p)$.

For signing a message m the signee picks a random number $k \in \mathbb{Z}_q$ and calculates as his signature the pair $(r,\ s)$ with $r = (g^{k^{-1}} (\mathrm{mod}\, p)) \bmod q$ and $s = k(m + xr) \bmod p$.

In order to verify a signature the verifier computes $r' = ((g^{ms^{-1}} y^{rs^{-1}}) \bmod p) \bmod q$. The signature verifies if $r' = r$.

Threshold DSA

Threshold DSA was presented by Gennaro in (GJKR01). Similar to threshold RSA Shamir's secret sharing is applied. A trusted dealer selects the system parameters p, q, g and a random polynomial $f(x) = s + \alpha_1 x + \alpha_2 x^2 + \ldots + \alpha_{k-1} x^{k-1}$ over $\mathbb{Z}_q$, where $s = x$. For each participant P_i the trusted dealer computes the secret shares x_i with $x_i = f(id_i) \bmod q$.

If VSS should be applied, the trusted dealer publishes the verification values $V_k = g^{\alpha_k}$ for $k \in [0, t-1]$.

Using Joint Secret Sharing this process can be performed without a trusted dealer. In that case, $3t-2$ founding members are required.

To generate a threshold DSA signature for a requester R using $2t$ signees, the signees will first run two instances of Joint Secret Sharing, which is a variant of Pedersen's Secret Sharing (see Section 2.4.5.1). in order to generate individual shares k_i and a_i of the random numbers k and a. Then, the signees run two instances of Joint Zero Secret Sharing in order to generate shares b_i and c_i of the 0. Verification values are calculated and distributed in order to apply Verifyable Secret Sharing and ensuring that all k_i, a_i, b_i, and c_i are consistent.

Now each P_j computes $v_j = k_j a_j + b_j \bmod q$ and $w_j = g^{a_j} \bmod p$ and sends these values to the other signees. Each signee then computes locally $\mu = \sum_{i=1}^{2t} (k_i a_i + b_i) \prod_{j=1, j\neq i}^{2t} \frac{-x_j}{x_i - x_j} \bmod q$ and $\beta = g^a \bmod p = \prod_{i=0}^{t} w_i^{\prod_{j=1,j\neq i}^{2t} \frac{-x_j}{x_i - x_j}}$. With this they compute $r = \beta^{\mu^{-1}} \bmod p \bmod q$ and send the partial signatures $s_i = k_i(m + x_i r) + c_i \bmod q$ and r to the requester R.

The requester computes the $s = \sum_{i=1}^{2t} s_i \prod_{j=1, j\neq i}^{2t} \frac{-x_j}{x_i - x_j} \bmod q$.

Evaluation

Distributed key generation of DSA keys is much more efficient than of RSA keys. However, it is obvious that threshold DSA will generate significantly more traffic for signing a document then threshold RSA. In DSA, at least $2t$ signees have to participate, because two polynomials of degree $t-1$ are multiplied with each other. Therefore, the communication complexity between the signees is $O((2t)^2)$, where it is for RSA based signing $O(t)$. Thus, for p2p systems this trade off has to be evaluated carefully and depends on the number of required signees.

2.4.7.6 Threshold Schnorr

The Schnorr signature scheme (Sch91) is a variant of the El Gamal scheme, which is also being applied in DSA. The corresponding threshold scheme was presented in (GJKR03). It is more efficient than the DSA threshold scheme. However, the proof of security requires additive secret sharing rather then polynomial secret sharing. Thus, Schnorrs scheme does not meet the requirements of this dissertation.

2.4.7.7 Threshold BLS

A signature scheme based on the Computational Diffie-Hellman assumption was presented in (BLS01, BLS04) (BLS01, BLS04) by Boneh, Lynn, Shacham (BLS). The BLS-scheme produces half the length of DSA signatures. The corresponding threshold scheme was presented in (Bol03).

BLS Signature Scheme Basics

The BLS signature scheme is a short signature scheme and is based on Elliptic Curves and Pairing-Based Cryptography. This is based on the concept of bilinear maps with two groups G_1 and G_2 of prime order q. G_1 is generated by g. A private key is a number $x \in_R \mathbb{Z}_q^*$. The corresponding public key is given by $y = g^x$, an element in group G_1. Further, H is a hash function with $H : \{0,1\}^* \rightarrow G_1$ that maps binary strings to non-zero points in G_1.

The signature σ on a message m is $H(m)^x$ (in G_1).

In order to verify a signature it must be checked if (g, y, $H(m)$, σ) is a Diffie-Hellman quadruple. This is the case if $e(g, \sigma) = e(y, H(m))$, where $e : G_1 \times G_1 \rightarrow G_2$ is a bilinear mapping.

Threshold BLS

The Threshold BLS scheme presented in (Bol03) applies (GJKR99) (see Section 2.4.6.1) to generate the shares x_i of the private key x. For each x_i the validation values $B_i = g^{x_i}$ can be calculated in order to verify the shares.

To generate a signature for a message m, each of t participants P_i computes $\sigma_i = H(m)^{x_i}$ and sends it to the requester. The requester can verify the received partial signatures by checking the Decisional Diffie-Hellman [6] $V_{DDH}(g, B_i, H(m), \sigma_i) = 1$. The requester then combines the partial signatures to the final signature by computing $\sigma = \prod_{i=1}^{t}(\sigma_i^{L_i})$, where L_i is the Lagrange coefficient.

Summary

Threshold BLS is another threshold cryptography scheme that can be applied to p2p systems. It has several advantages compared to RSA-based and DSA-based threshold schemes. Like in RSA the signing process has a message complexity of $O(t)$; thus it is significantly more efficient than threshold DSA during the signing process. Further, BLS keys are based on simple random numbers; thus the distributed key generation is significantly more efficient than for RSA keys. However, there is a drawback. Another advantage is its short signatures. Therefore, its signatures create less traffic overhead RSA and DSA signatures. Threshold BLS is computational intensive (cf. (STY07)), i.e., it is not suited for small devices with little processing power.

2.4.8 Proactive Secret Sharing

Proactive Secret Sharing (PSS) schemes were introduced to protect long-lived shares of cryptographic keys. Threshold cryptography increases the availability and the security of a secret key, but it is still

[6] see (Bon98)

possible to attack enough shareholders to reconstruct the secret. Shares can become corrupt, so there is no guarantee that the secret can be reconstructed.

PSS solves the most problems for long-lived shares of keys. The principle of PSS is to introduce time periods. In each time period the shares of the shareholders are updated, and potential attackers are not able reconstruct the secret with shares from different time periods. Thus, the time periods have to be adapted, so that an attacker cannot gain access to more than k shares. Furthermore, PSS offers the possibility of identifying and recovering corrupted shares or shareholders.

In order to further explain the principles of PSS, the scheme of Herzberg et al. (HJKY95) is illustrated in more detail. This PSS scheme is modified by the URSA scheme (see Section 2.4.7.4) in order to work with RSA signatures.

2.4.8.1 Herzberg's Proactive Secret Scheme

Initialisation Phase

In initialisation phase a dealer D is needed to distributes the shares of the secret s. Therefore, the dealer follows Feldman's verifiable secret sharing scheme (Section C.2.2.1). If the initialisation phase is performed correctly, each shareholder has a valid share and no accusations against the dealer are broadcast. If an accusation is broadcast, some responses are required as described later in this section (2.4.8.1).

Reconstruction Phase

The reconstruction phase is similar to the phase presented in Section 2.4.4.1. k shares are sent directly to the secret requester, who can now reconstruct the secret. This approach can be adjusted to suit RSA secret sharing (see Section 2.4.8.2).

Update Phase

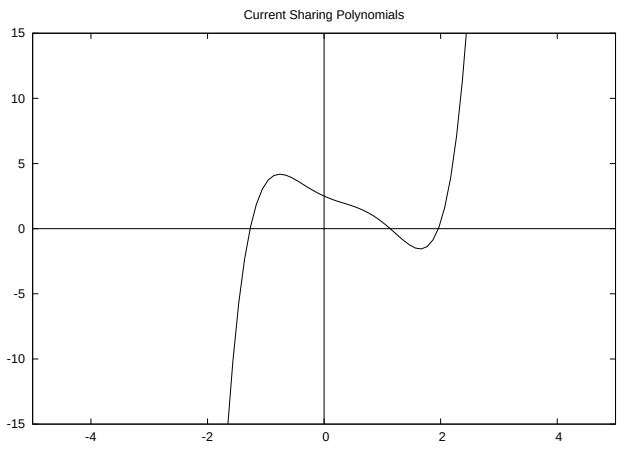
After initialisation phase, all shares are distributed to the different shareholder. To increase the security of these shares, they are updated at predefined time intervals. At the end of such an update phase, each shareholder has a new share, but the secret s remains the same. To successful update all shares, all shareholders have to participate in the update process. The following steps are performed in the update phase between time period t and $t+1$:

- Each shareholder P_i (i=1..n) picks $k-1$ random numbers and creates a polynomial $d_i(x)$ of degree k-1, where $d(0) = 0$, following Equation 2.12.

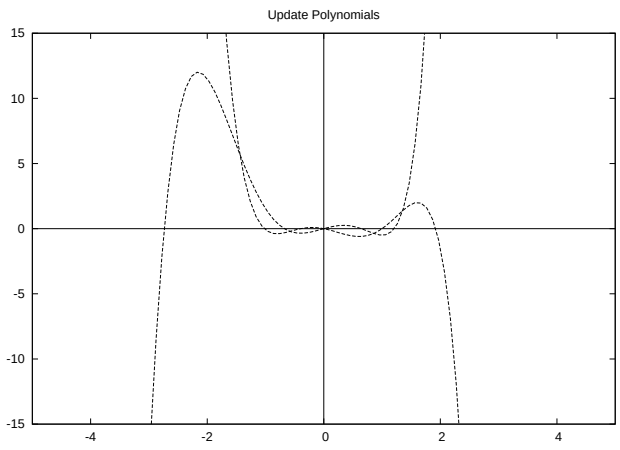
$$d_i(x) = d_1 x + d_2 x^2 + \ldots + d_{k-1} x^{k-1} = \sum_{j=1}^{k-1} d_j x^j \bmod q \tag{2.12}$$

- P_i creates the updated shares $u_{ij} = d_i(j)$ and computes the verification values $g^{u_{ij}}$. The share u_{ij} and the verification value are sent secretly to shareholder P_j. The verification values of the coefficients of the update function ($g^{d_i}, i = \{1 \ldots k-1\}$) are broadcast to all shareholders.

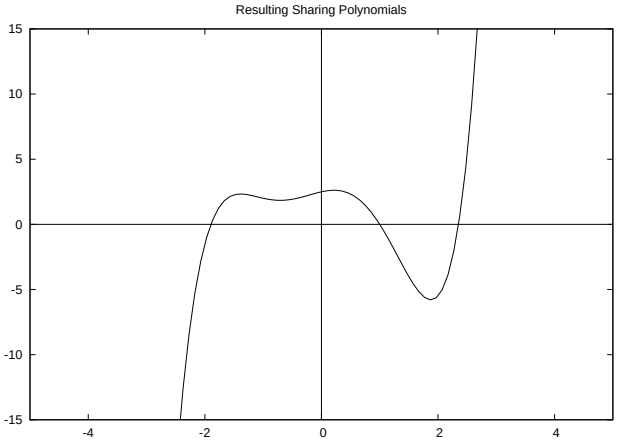
- If shareholder P_i has received the updated shares and verification values from all shareholders (that are participating the update process), it checks the correctness of the shares (using Equation C.13). If some failures occur, shareholder P_i broadcasts an accusation against the corresponding shareholder. If the equation holds, shareholder P_i adds the received shares to its old share $s_i^{(t)}$ and obtains the new share $s_i^{(t+1)}$ (Equation 2.13).



(a) Current sharing polynomial



(b) Update polynomials



(c) Resulting sharing polynomial

Figure 2.7: Graphical representation of the update phase

$$s_i^{(t+1)} = s_i^{(t)} + \sum_{j=1}^{n} u_{ji} \tag{2.13}$$

- To protect the secret, all received updated shares and old shares are erased completely.

A graphical representation of these steps is shown in figure 2.7. The graph in the upper left represents the function used to reconstruct the secret s in time period t. The graph in the upper right shows the update functions of two shareholders used to compute new shares without changing the secret s. Thus these functions are zero in the point of origin. If these two functions are added to the first function, the third function is created (shown in the graph in the lower middle).

Recovery Phase

In recovery phase, corrupted shareholders are identified and recovered. This is necessary to provide the continuous number of shareholders that have to be available to reconstruct the secret s. Otherwise, an attacker can consecutively destroy or corrupt the shares of the shareholder to inhibit the reconstruction of the secret.

To identify and remember a corrupted share, each shareholder generates a list of good and bad shareholders. This list is generated after the initialisation or update phase and contains all malicious shareholders identified after resolving the accusations. Additionally, the shares of all shareholders not involved in the last update phase have to be recovered.

In addition to the obviously malicious shareholders, the shareholders attacked in the last time period have to be identified. Herbage et al. proposed that all shareholders store the public verification values of the shares ($g^{s_i}, i \in \{1 \dots n\}$) broadcast during initialisation phase. If the shares are updated, these verification values will be updated too by using Equation 2.14.

$$g^{s_i^{(t+1)}} = g^{s_i^{(t)}} \prod_{j=1}^{n} g^{u_{ji}} \tag{2.14}$$

whereas $s_i^{(t+1)}$ is the share from time period $t+1$ and u_{ji} is the updated value that is added to the share $s_i^{(t)}$ by shareholder P_i.

To identify a malicious shareholder, each shareholder broadcasts the verification value of its actual share. Then all shareholders can decide by majority count which shares are valid. This results in a list of malicious shareholders that is the same for each shareholder.

To recover the corrupted shares the share recovery protocol is performed. In the following, $\mathcal{B}$ indicates a set of malicious or bad shareholders that are identified by the correct shareholders located in the set $\mathcal{A}$ ($\mathcal{A} \cap \mathcal{B} =$). At least k correct shareholders are needed to recover the secret and the following steps must be performed.

- Each shareholder $P_i \in \mathcal{A}$ chooses for each malicious shareholder $B_k \in \mathcal{B}$ a random function $d(x)$ of degree $k-1$ with $d(k) = 0$. This can be achieved with Equation 2.15

$$d_k(x) = d_1 x + d_2 x^2 + \ldots + d_{k-1} x^{k-1} - \sum_{i=1}^{k-1} d_i k^i \bmod q \tag{2.15}$$

  whereas k is the variable of the malicious shareholder B_k used to generate its share and d_i ($i = \{0, \dots, k\}$) are the coefficients of function $d_k(x)$.

- Then P_i send the value $u_{k:ij} = d_k(j)$ secretly to each shareholders $P_j \in \mathcal{A}$.

- P_j uses the received shares $u_{k:ij}$ from each $P_i \in \mathcal{A}$ to compute a share for B_k using Equation 2.16.

$$s_{jk} = s_j + \sum_{i \in A} u_{k:ij} \tag{2.16}$$

  where s_j is the share of shareholder P_j and $u_{k:ij}$ is the recovery value generated by shareholders $P_i \in A$ for P_j. The resulting share s_{jk} is transmitted secretly from P_j to B_k.

- The malicious shareholder $B_k \in B$ receives the shares s_{jk} from at least k shareholder $P_j \in A$ and uses Lagrange interpolation to recover its share (see Equation 2.17)

$$s_k = \sum_{j=1}^{k} s_{jk} \prod_{i=1; i \neq j}^{k} \frac{k-i}{j-i} \tag{2.17}$$

Resolving Accusations

If during initialisation, update, or reconstruction phase one or more accusations are broadcasted, the malicious participants must be identified. Two possibilities exist:

- Discard all shares from the accused shareholder
- Try to identify the malicious participant (the accused shareholder or the accuser)

In the first case, all shareholders ignore the messages send by accused shareholders and the shares of the shareholder are reconstructed like in reconstruction phase. But this offers an attacker the possibility to accuse any shareholder and to make the reconstruction of a secret impossible. In the second case the accused shareholder defends itself by showing that its share is correct. Therefore, the accused shareholder publishes the accused share and maybe additional information, to demonstrate that this is the same share that was sent to the accuser. Now, all shareholders can check the correctness of this share. If the shareholders decide that the published share is correct, the accused shareholder defended itself successfully and the accuser is marked as bad and thus is added to a list of bad shareholders ($\mathcal{B}$). Then the bad shareholders are recovered in the recovery phase.

In the following proactive secret sharing for URSA and for Threshold BLS are discussed.

2.4.8.2 Proactive Secret Sharing Using the URSA Scheme

Update Phase

The URSA scheme (LKZ+04) uses the previously presented update phase of (HJKY95). To differentiate between the various updated shares, version numbers are introduced.

Recovery Phase

The recovery phase is based, in part, on (HJKY95) (see Section 2.4.8.1). To recover a lost share k shareholders cooperate and use Lagrange interpolation (see Equation 2.18). Whereas $s_{i:r}$ is the share that is sent from shareholder P_i to P_r.

$$s_r = \sum_{i=1}^{k} s_i \prod_{j=1;j\neq i}^{k} \frac{r-j}{i-j} = \sum_{i=1}^{k} s_{i:r} \tag{2.18}$$

To avoid a situation where shareholder P_r is not able to compute the distributed secret (the private key), the k shareholder communicate in advance and add random numbers to their shares. This is the difference from (HJKY95). Herzberg proposed generating recovery functions that are zero at the index of the shareholder (see Section 2.4.8.1). (LKZ+04) proposed using random numbers. Each shareholder communicates with the $k-1$ other shareholders and they exchange random numbers. The shareholder with the higher ID treats the random number as positive, the shareholder with the lower ID as negative. So each shareholder gets $k-1$ random numbers and sums them to its own share $s_{i:r}$. These shares are sent to P_r that uses Equation 2.17 modulo N to recover its share.

Verification of Shares

In (LKZ+04) it is written that the verification methods proposed by (Fel87) or (Ped91b) can be used. However, one important limitation exists, because of the fact that the shares are computed modulo N. Let $f(x) = f_0 + f_1 x + \ldots + f_{k-1}x^{k-1} \bmod (N)$ be the function used to generate update shares $s_i = f(i)$. The verification values of this function are $g^{f_0}, g^{f_1}, \ldots, g^{f_{k-1}}$ and the verification values of the shares are g^{s_i}. With regard to Equation C.13, the share s_i can be verified by checking $g^{s_i} \bmod N = g^{f(i)} \bmod N = \prod_{j=0}^{k-1}(g^{f_j})^{i^j} \bmod N$. This is not applicable to the scheme presented above, since $f_i i^j \neq f_i i^j \bmod N$. Especially $g^{s_i} \bmod N = g^{\sum_{j=0}^{k-1} f_j i^j \bmod N} \bmod N \neq g^{\sum_{j=0}^{k-1} f_j i^j} \bmod N$.

Security

For the URSA-scheme, the potential for an attack exists when proactive secret sharing is applied, because there is information leakage during the update phase. The attack is described in (JSY04, Sax06). The attack is successful if an adversary compromises any set of at most t members in the update group in every update round. When compromised, the adversary learns the corresponding private key share and can control the participants' behaviour during the update phase.

The attack can be applied if the attacker has successful attacked $k-1$ shareholder (in set $\mathcal{B}$) and is able to compute the value $S = \sum_{i \in \mathcal{B}} s_i$. Then the attacker requests a signature. By constructing the final signature, the attacker gets the value of α (c.p. Equation 2.11) and learns if the value of the private key d is greater or less than $S \bmod N$. This is because $S < \alpha N$ if and only if $d < S \bmod N$. Otherwise $d \geq S \bmod N$ if and only if $S \geq \alpha N$. By choosing an appropriate update function, the shareholder can control the value of S and can isolate the value of the secret. Therefore, the attacker has to be the last shareholder that distributes update shares. Based on the update shares received from other shareholder he can choose his own update values and generate a value of S at will.

At first blush, this attack seems to be very dangerous. But in practise, it is very unlikely that the same group of $k-1$ attacked shareholders will be chosen multiple times to generate a signature in cooperation with only one good shareholder. However, to avoid this attack or problem, one has to change the group that generates the signatures randomly. Additionally, the threshold can be increased so that we have an (b, k, n) threshold scheme with $b < k$.

In (Sax06) the efficiency analysis of the attack shows that the private key can be compromised after 163 rounds for 1024-bit N, $e = 65537$ and $t = 7$, if (1) t members of the update group are corrupted and one of them speaks last during the update phase; and if (2) in every update phase some h honest participants participate in a signature process.

In (Sax06) a solution to the information leakage is presented. However, this solution relies on additive secret sharing, which is why it cannot be applied for autonomous distributed systems with a dynamic number of participants.

Summary

2.4.8.3 Proactive BLS-Scheme

The BLS-scheme satisfies the requirements for secure proactive secret sharing stated in (HJJK97) and (HJKY95), which have been described in Section 2.4.8.1. Therefore, these schemes can be applied. Threshold BLS applies Pedersen's Verifiable Secret Sharing (Section 2.4.5.1) within the proactive secret sharing phases.

Summary

Proactive Threshold BLS offers all mechanisms that are desired for an application in p2p systems. These are distributed key generation and verification of update shares. Further the signing process of Threshold BLS is efficient. Accordingly, it is the best choice if signatures are to be created in a fully distributed manner in a p2p system.

2.4.9 Conclusion Security Mechanisms for Distributed Systems

The discussion of security mechanisms for distributed systems presented the basic concepts of secret sharing, distributed key generation, threshold cryptography, and proactive secret sharing.

As a basic requirement, it can be concluded that for autonomous distributed systems with a dynamic number of participants, it must be possible to generate additional shares even after the initialisation phase. Thus, additive secret sharing cannot be applied, and any applied scheme should be based on Shamir's polynomial secret sharing.

Further, as it must be taken into account that if an adversary tries to get knowledge of the shared private key, proactive secret sharing should be applied. Three different kinds of proactive secret sharing schemes exist:

RSA-based proactive threshold schemes are the most discussed in the literature. However, for large systems only the URSA-scheme can be applied. The URSA has security hole that would require generating a new key pair after a specific number of update rounds. The distributed key generation is very traffic intensive due to the requirement to generate prime numbers in a distributed way. Its advantage is the efficient signature protocol that requires only a linear number of messages with growing threshold t.

DSA-based schemes do not have these disadvantages. The distributed key generation is much more efficient than distributed RSA-key generation, as no prime numbers need to be generated. However, its disadvantage is that the signature protocol requires several broadcasting phases. Further, $2t$ participants are required for signing a message.

The most efficient scheme seems to be the BLS-threshold scheme. The key pair can be initialised using (GJKR99, GJKR07). Furthermore, the signature protocol requires a linear number of messages with growing threshold t. Also, its security has been proved to be without significant faults.

Verification of shares and partial signatures is an important feature of proactive secret sharing. For distributed generation of a key pair, the participants have to broadcast individual verification values for each participant to all participants. URSA does not provide this mechanism, however Threshold DSA and Threshold BLS do. An evaluation of the BLS-threshold scheme in terms of produced traffic and computational requirements is presented in (STY03). This shows that the computational complexity of the BLS-scheme is higher then for the other schemes.

When applying proactive secret sharing to autonomous distributed systems, boot strapping a single server for generating a system-wide key pair should be avoided. Therefore, proactive threshold schemes that allow a distributed key generation are required. For RSA-based threshold systems the distributed key generation presented in (BF01) and optimised for efficiency in (FMY98) can be applied. This was already discussed in Section 2.4.6.2. For discrete-log based schemes like DSA and BLS the method presented in (GJKR99, GJKR07) should be applied, as it represents the most actual findings. This was discussed in 2.4.6.1.

In conclusion, for applying a proactive secret sharing scheme to autonomous distributed systems, the BLS-scheme is the best suited scheme, as it offers all characteristics desired for an application in p2p systems. It scales to an unlimited number of peers, it provides verifiable secret sharing, and it allows efficient distributed generation of keys. However, should proactive secret sharing be applied on devices with low computational power, the URSA-based scheme can be more appropriate. In that case, it is important to pay close attention to the constraint that after a specific number of update rounds a new key must be generated.

2.5 Summary

To fulfil the goal of demonstrating the feasibility of a fully distributed accounting scheme for p2p systems with intrinsic automatic cooperation control, this chapter has reviewed the related work.

The first section gives the relevant definitions in context of this dissertation. An overview on the different definitions of p2p systems is given and the definition of Steinmetz and Wehrle (SW05a) is selected as the most relevant for this dissertation. Accounting in information systems is defined as the process of tracing relevant IS activities to a responsible source. Cooperation control is understood as a three part process consisting of definition of rules, control of rules, and enforcement of rules. Finally, security and trustworthiness are circumscribed. A trustworthy system is designed with good intentions, has the ability to perform the functionality it was designed for, and is secure. Secure refers to information security and data protection.

The second section reviews accounting functionality in computer networks, apart from p2p systems. All solutions for accounting in centralised systems, decentralised systems, in the Grid, as well as micro-payment schemes apply a central trusted entity. Therefore, these solutions cannot be applied to p2p systems.

The third section reviews mechanisms presented in the context of p2p systems that could be applied to p2p systems. First, a classification is given for accounting in p2p systems. It structures the solution space according to the type of information collection, the information storage location, and the type of issuer for systems that use an issuing process. All relevant works are classified and discussed in detail. It was concluded that the most efficient solution for trusted accounting in p2p systems uses local information storage, because this allows a message complexity of $O(1)$ during transactions. However, there was no existing trustworthy solution found that applies this design decision. Karma (VCS03) is closest to the goal of this dissertation. However, it relies completely on the availability of bank sets and the correctness of information the bank sets store. There is no analysis presented that concludes the required bank set size in order to guarantee availability and trustworthiness of information.

The final section reviews the related work in the field of security mechanisms that can be applied in decentralised autonomous systems. Threshold cryptography is an especially well suited mechanism that allows delegating the signing of a document with a private key to a group of peers. Each peer possesses a share of the private key and no peer possesses the knowledge of the complete key. Using proactive secret sharing, the secrecy of the shared key can be achieved over time. However, the intensive study of threshold cryptography mechanisms revealed that most schemes cannot be applied to p2p systems due to various limitations. Using additive sharing all key shares have to be created when the key is created and no additional key shares can be created afterwards. Using Shamir's Secret Sharing this limitation does not exist, however most solutions cannot be applied in large systems. Finally, two threshold schemes could be identified that fulfil these requirements. Threshold BLS presented in (Bol03) is a short signature scheme, however it has computationally high requirements. It is not yet suited for application in systems with low computational capabilities. Therefore, the URSA threshold scheme (LKZ$^+$04) was selected as a second choice. However, it has a security leak and requires the generation of new shares after 160 update phases. Furthermore, it does not support verifiable secret sharing yet.

After giving the relevant definitions in context of this dissertation, reviewing the related work, and laying the technical foundations for establishing a distributed basis of trust in p2p systems using threshold cryptography, all of the basics for designing a fully decentralised and trusted accounting mechanism with intrinsic cooperation control for p2p systems are worked out. The following chapter presents the framework, system architecture, and basic protocols of such a system, the novel token-based accounting scheme.

3 Token-Based Accounting Principles and Architecture

This dissertation presents the token-based accounting scheme that enables accounting with intrinsic cooperation control in autonomous distributed systems. The requirements for accounting and cooperation control have been stated in Chapter 1.

In this chapter the basic framework and system architecture of the token-based accounting scheme are derived from the identified requirements. All fundamental design decisions are explained. The result is a framework that allows efficient collection of accounting information and *effectively control cooperation using the accounting information*. The token-based accounting scheme consists of four closely interlinked building blocks. Removing one block would compromise the provided functionalities or characteristics of the other blocks.

After the framework is derived the token-based accounting schemes architecture is described by elaborating each of the four building blocks: token structure, transaction protocols, token aggregation, and detection of double spending. Token aggregation implements *fully decentralised token creation and administration*. Tokens are used to control cooperation, therefore an efficient *decentralised control of double spending* is provided by the system. Finally, a *trustworthy transaction* is offered to peers, which removes the incentive to defraud the partner in a transaction.

3.1 Assumption

For designing the token-based accounting scheme the following assumption have been taken into consideration:

User Identification

The token-based accounting scheme requires that actions can be clearly attributed to users. For the accounting system presented here, it is assumed that each user can clearly be authenticated using its private/public key pair.

Certification Authority

In order to authenticate a user with its key pair, functionalities like those offered by a public-key infrastructure (PKI) must be available in the system. This will clearly attribute a key pair to a user. There is no restriction requiring all users to use the same PKI; each peer can use the PKI of its choice. The architecture of the PKI is beyond the scope of the dissertation, as it does not further affect the accounting system's design.

Peer Identification

The token-based accounting scheme assumes that users can clearly be identified through a permanent identification (ID) in the overlay network. A permanent ID can be obtained as hash value of a users public key, like in (Wal02, JXT04).

Permanent peer identification in combination with user identification issued by a certification authority eliminates potential Sybil attacks (Dou02) or Whitewashing attacks[1] in the p2p system.

[1] see Appendix C.1.3 for a definition.

Reputation Mechanism

The token-based accounting scheme does not rely on a central trusted entity. Therefore, in order to detect dishonest behaviour, all system participants must observe other peers behaviour. That is, if peers are interacting with other peers and detect dishonest behaviour, there must be a mechanism to report and to punish such behaviour.

It is assumed that there exists a decentralised reputation system, where peers can report dishonest, fraudulent behaviour, which cannot be detected and prevented by other technical means. It is assumed that each peer has a reputation value that reflects the honesty of its past actions in the system.

Several possible solutions for such a decentralised reputation mechanism have been proposed. See, e.g., (YS00, KSGM03, XL04, DA06). The design of the applied decentralised reputation mechanism is beyond the scope of this dissertation.

These assumptions do not restrict the design of the accounting system. Rather, these assumptions form the basis the accounting system is built on and are required in order to clearly focus the research on the design and the evaluation of an accounting system for autonomous distributed systems.

3.2 Design Decisions and Resulting Framework

As the related work in Section 2.2 shows, to design an accounting scheme for autonomous distributed systems is a complex task, as a central trusted entity is missing. In the following, the basic system design is derived from the design goals stated in Chapter 1.

3.2.1 Accounting Objects

As described in Chapter 1, the primary goals of the proposed system are

1. to globally collect accounting information about resources and services provided and consumed by the peers in a p2p system.
2. to enforce norms about cooperation based on the collected information by constraining access to resources and services accounted for.

Required System Characteristics

These goals imply five characteristics the system requires to achieve:

- The system *collects receipts* of the transactions the system accounts for. These collected receipts will be referred to as accounting information.
- The accounting information is *collected system-wide* and *per peer*.
- When *accessing* the collected accounting information it is available in a *complete*, *correct* and *prompt* manner.
- In order to enforce norms of cooperation, there is a *mechanism to deny access to services or resources* for entities that do not comply with the norms.
- There is a *clear, apparent, and unforgeable flag* showing, if a peer is permitted to access services or resources.

Another general requirement is the trustworthiness of the accounting scheme. Due to the assumptions the scheme is built on, Sybil attacks (Dou02) as well as Whitewashing attacks are prevented. When designing the system potential cheating and collusion of peers has to be taken into special consideration.

Hereafter, peers not behaving according to the rules will be called “misbehaving peers”.

The complexities of the researched solution are completely different than in client-server oriented systems with a central trusted entity that administers accounting information. How the remaining characteristics can be realised in a distributed autonomous system will now be addressed.

Accounting Receipts

The first characteristic is obvious to achieve. A receipt must be created that contains the required accounting information about each transaction, be it a service or resource usage. The information authentication, integrity and non-repudiation must be ensured.

As an example, peer A downloads a media file from peer B. Here, A has to certify that it received a service from B in form of a receipt. Both transaction partners should state their agreement to the information contained in the receipt.

Basic Cooperation Rule

In order to coordinate resource and service usage, a balance between offered and demanded resources and services must be achieved. Demand for free-of-charge resources and services will always exist. In contrast, free-of-charge resources and services will only be offered by altruistic system entities, which must be assumed to be the distinct minority of peers in the system. Thus, a balance between offer and demand can only be achieved if resource and services usage requires to reciprocation.

In conclusion, a basic behaviour rule to be supported by the accounting system is that in order to use accounted resources or services, a peer also has to provide resources and services.

Enforcing Cooperation in Distributed Autonomous Systems

In distributed autonomous systems, it is hard to enforce cooperation, as no central entity exists that can deny access to the distributed resources or services. The problem can best be observed from the opposite direction. Each peer that wishes to participate in the distributed autonomous system must have the permission to do so. This permission can be bound to an object. Only if the peer is in possession of such an object, can it use resources and services.

There are two alternatives for such a permission object. In the first alternative, a permanent permission object could be issued to peers. When a peer is not permitted to consume services or resources anymore, the permission will be revoked, and the peer will be black-listed. This implies that for each transaction in the system, the service provider first must check the black-list. The second alternative issues a permission object to peers that can be used only once. Here, the behaviour enforcement is more fine grained. However, there must be controls ensuing that permission objects are not used more than once. This is called *Double Spending*.

An accounting system requires a receipt per transaction. This could be combined with a corresponding permission object per transaction. Therefore, the second alternative presented above, using a permission object with onetimeuse is the preferred solution.

As a permission object should be used only once, there must be a way to use it up. The solution is to design it as “fading”. That is, if the permission object is used, it “fades” and cannot be used again. But it can be renewed if the peer acted according to the rules.

The result of these considerations is to introduce objects in the distributed autonomous system that represent the things that should be accounted for, i.e. they represent the right to use the services and resources that are accounted for. These objects should be scarce in order to enable behaviour rule enforcement. From this it follows that in order to ensure the scarceness of the permission object they must be issued by some decentralised process. This process must be governed by rules about when permission objects should be issued. The compliance with the rules must be checked by the system’s peers. They have a motivation to do so because, as previously described, it is assumed that the system

entities have an interest in a system that accounts for resource and service usage. Otherwise, they would move to a different system.

Tokens as Accounting Objects and Permission Objects

The accounting system for autonomous distributed systems with the described goals requires two types of objects: objects that hold accounting information and permission objects. One type of objects is required before a transaction takes place (permission), the other type is required after the transaction took place (accounting). Therefore, both objects can be combined into one object that is used throughout all phases of the transaction. An advantage is that identity information must be proved only once in a transaction, because both type of objects require that information.

Objects that hold accounting information typically are receipts. As the permission objects must be issued, the accounting system requires pre-transaction issued receipts. This special type of object is called a ***Token*** throughout this dissertation.

Renewing Permission Objects

Tokens are "fading" permission objects. A peer will loose tokens when consuming resources or services. This will be called ***spending*** tokens. However, when behaving according to the rules a peer should get the tokens "re-newed". "Re-newing" tokens means, new tokens should be issued to the peer. According to the basic behaviour rule tokens should be issued when providing resources or services. As proof for provisioning also tokens are used, as they are the receipts. Thus, system must support a mechanism where peers can swap "receipt-tokens" against new "permission-tokens".

Basic Accounting Scheme Concept

In conclusion, every participant in a system applying the accounting mechanism has a limited number of tokens they can use in transactions. The service provider keeps the accounting records about transactions. A participant who runs out of issued receipts is not allowed to consume any further services. This mechanism enables the enforcement rules. Participants will collect tokens when providing accounted resources or services. These "token-receipts" they can swap for new tokens. This mechanism enables to coordinate resource and service usage in the distributed autonomous system.

As example, Peer A requests a media file from Peer B. When B uploads the media file to A, A will take one of its tokens and fill in accounting information required for that service. Then it will send the token over to B. B receives a used token and will swap it for a new token. Thus, A spent a token and B earned a token. B keeps the accounting record about the transaction stored in the used token.

As a consequence, potential forgery and double spending of tokens must be addressed in the design of the accounting scheme [2].

Figure 3.1 depicts the concept. Note, that the location where new tokens and used tokens are stored has not yet been decided. Furthermore, note that the decentralised reputation system is separate from the token-based accounting scheme (see Section 3.1) and its design is beyond the scope of this dissertation.

3.2.2 Issuing Tokens

When designing the process for issuing tokens, two basic design decisions must be made. Which entities, i.e. peers, are responsible for issuing tokens, and how can a correctly issued token be signed.

[2] see Appendix C.1.3 for definitions.

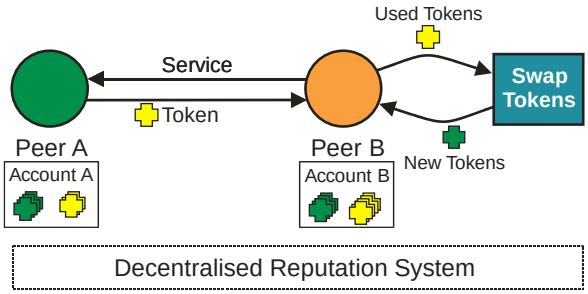


Figure 3.1: Basic token-based accounting concept

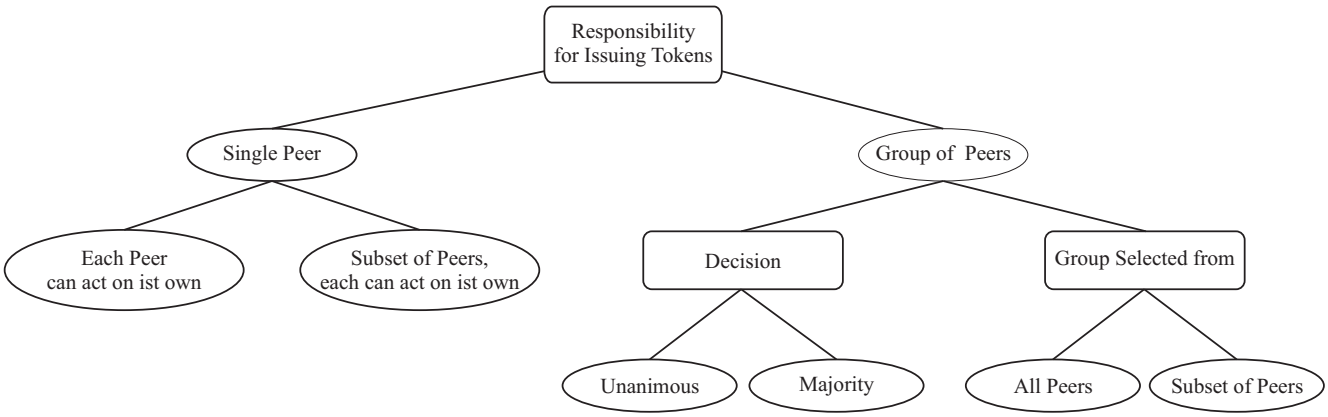


Figure 3.2: Solution space for token issuing responsibility

Responsibility for Issuing Tokens

As described above, each token must be issued. However, in a distributed autonomous system there exists no trusted entity that could take on this functionality. Therefore, a trustworthy solution must be found that is feasible within the peer-to-peer paradigm. Figure 3.2 depicts the solution space.

If a single peer should be permitted to issue tokens, either each peer can have this permission or only selected peers. The selection can be performed by trust. However, for the token-based accounting scheme, it cannot be assumed that a single peer can be trusted to issue the tokens. Token-issuing is a too important a task. Therefore, the second alternative, a group of peers issuing tokens, must be selected.

If a group of peers shall issue tokens, this could mean all system peers or a sub-group of all system peers. If all peers potentially can take part in token issuing, that means that potentially fraudulent peers take part in this process. This must be minimised. As a reputation mechanism is assumed to exist that provides an assessment of the peers' trustworthiness, a sub-group consisting of the most trustworthy peers will be the better design choice.

If a group of peers has to make a decision if tokens should be issued or not, this decision can be based on different forms of majority or on unanimity. Unanimity has the advantage that in a decision group, only one honest peer is required to avoid issuing tokens against the accounting scheme's rules. The disadvantage is that a single malicious peer can prevent tokens from being issued although this would not violate the accounting scheme's rules. If such malicious behaviour is detectable it can be reported to the reputation mechanism. Then, the token issuing could be repeated with a different group of peers. The trustworthiness of the token-based accounting scheme has priority the concern about tokens being wrongly withheld. Therefore, a group of peers should issues tokens based on a unanimous decision. This is the most trustworthy solution in order to avoid fraud. The group of peer deciding about issuing tokens is called ***quorum***.

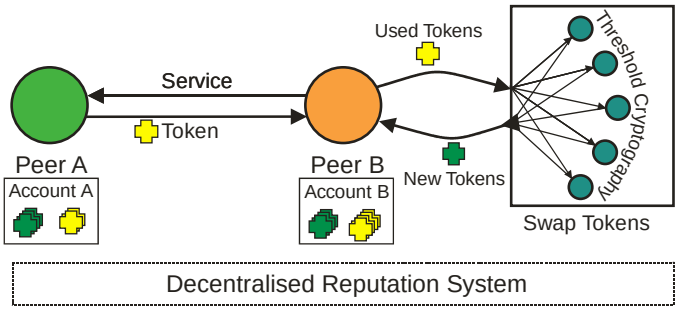


Figure 3.3: Preliminary token-based accounting framework with threshold cryptography

Sign for Valid Tokens

Typically, in order to state a documents authentication, integrity, and non-repudiation it is signed with some kind of signature. In the token-based accounting scheme, issuing a token means creating a valid signature under a token.

There exist three alternatives for this signature:

1. In this option, there would exist a scheme's private/public key pair. Each of the quorum peers has the knowledge of the scheme's key and can sign tokens with it. A token requires at least t signatures of quorum peers with the system key. However, the signatures of the different quorum peers cannot be distinguished. Therefore, the signatures could also be created by a single peer. Accordingly, this solution cannot be applied.

2. The quorum peers sign tokens with their private key. If a token is signed with at least t signatures, it is valid. Here, if a peer plans to defraud the system, the quorum peers could be exchanged for others until a quorum consists only of peers who are willing to defraud the system. Furthermore, the token size would increases due to the number of signatures. This would increase the total traffic introduced by the accounting scheme.

3. Proactive threshold cryptography is used with a system-wide key that is shared among the quorum peers. The quorum that signs a token must be pre-selected due to the cryptographic requirements. More specifically, the Lagrange coefficient avoids exchanging peers in the quorum , which aggravates collusion. Only one signature is created under a token. That reduces the traffic introduced by the token-based accounting schemes. On the other hand, additional traffic is introduced for key management.

In conclusion, the third alternative is the most trustworthy one. Therefore, it is selected for the token-based accounting scheme.

From the discussion of the related work, the BLS-threshold scheme (BLS01, BLS04, Bol03) is best suited for the token-based accounting scheme. However, this scheme is computationally the most demanding, as (STY03) shows. Therefore, the URSA-threshold scheme could also be applied, when small devices with low computational power may be involved in the signing process. However, close attention must be paid to the security limitations of the proactive URSA protocol. After a specific number of update rounds, a new key must be generated (see (Sax06)).

Fortunately, for creating signatures using either the BLS-threshold scheme or the URSA-threshold scheme the required message flow is the same.

Figure 3.3 shows the includes the resulting issuing process into the basic concept of the token-based accounting scheme.

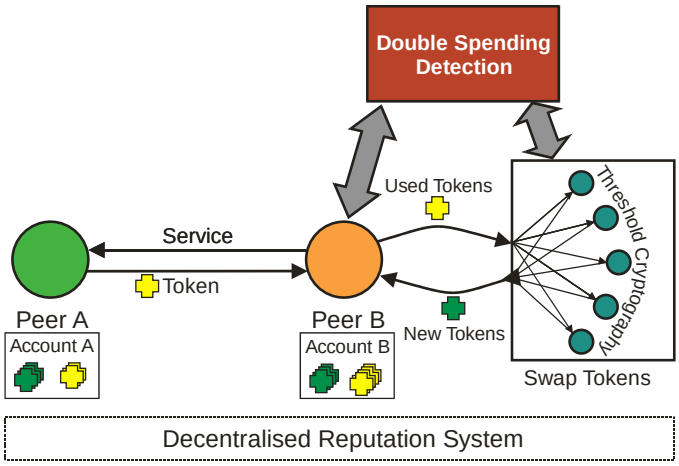


Figure 3.4: Extended token-based accounting framework

3.2.3 Location of Token Storage

Tokens are spent by peers and spent tokens can be swapped for new tokens. Accordingly, there are two types of tokens, new tokens and spent tokens. A storage location must be determined for both types.

There exist two alternatives of where tokens could be stored. Either token can be stored locally at the peer owning the tokens, or remotely with other peers. The object where the tokens are stored will be referred to as an ***account***.

If accounts are stored with other peers, this can have the advantage of being housed with peers who might not have an incentive to falsify account information. However, the disadvantage is that in each transaction, the account holding peers are involved and therefore must be available. This introduces additional traffic. Furthermore, account information must not be lost, otherwise the account owning peer will not be able to perform any transactions.

If accounts are stored locally at the owner peer it must be ensured that the account information is secure. That is, a peer has a direct incentive to falsify the account information for its own advantage. Thus, token integrity must be ensured, and forgery of tokens must be prevented. If these requirements can be fulfilled, local storage is to be preferred, because compared to account storage at remote peers, network traffic is lower and the availability of accounts is not an issue. Due to the design decision to issue tokens, for new tokens, information integrity and security against forgery is given. Therefore, new tokens can be stored locally at the owner peers. Spent tokens contain additional accounting information. For this information integrity must also be ensured. This can be done, if an owner peer signs the accounting information using its private key before spending it. As this procedure is required for an accounting system, spent tokens can also be stored locally at the receiving peer.

If tokens are stored locally, a further issue is double spending of tokens. Peers could copy tokens and use the same token repeatedly for consuming resources or services. If tokens are stored locally, there must be a mechanism to identify tokens clearly and to detect if a token was used more then once. In the system design selected so far, the best option for controlling how often a token was spent seems to be the time when spent tokens are swapped for new tokens.

3.2.4 Double Spending Detection

The token-based accounting scheme requires that tokens can be used only once. In order to detect if a tokens was used more then once, each token must possess a unique identity. There must also be an instance where it is recorded which tokens have been used. In distributed autonomous system this

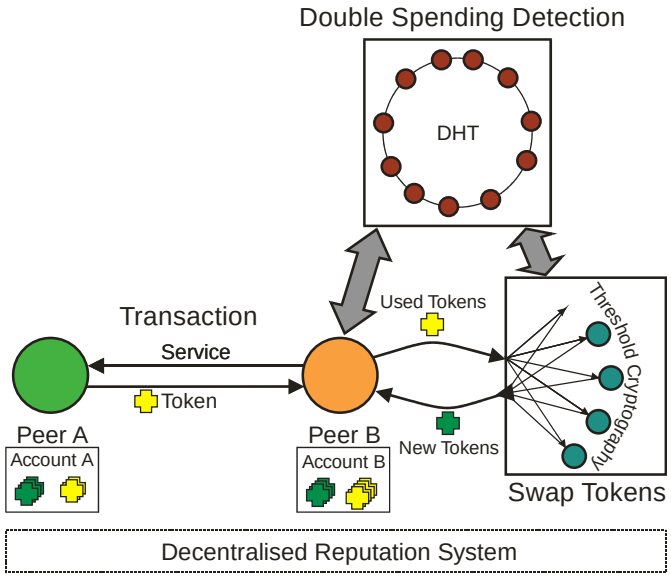


Figure 3.5: Complete token-based accounting scheme framework

instance is a distributed index. In peer-to-peer systems an efficient way to implement a distributed index is a distributed hash table (DHT).

There are two alternatives, as to how double spending can be detected. A DHT is built among the trusted peers or among all peers. If the DHT is built among the trusted peers, double spending can be detected when used tokens are swapped for new ones. When the DHT is built among all peers, it is also possible to store and retrieve information during a transaction. As this solution offers more freedom, it is selected.

3.2.5 Resulting System Architecture Overview

The resulting design of the accounting scheme for distributed autonomous systems uses tokens to accomplish its goals. These goals are primarily to collect accounting information, and to enforce norms about cooperation system-wide. Figure 3.5 a shows the resulting framework of the token-based accounting scheme. Note that the decentralised reputation system is a separate mechanism and its design is beyond the scope of this dissertation.

Issuing Tokens

Tokens will be issued to individual peers using proactive threshold cryptography in a decentralised process. A sub-group of peers will be responsible for issuing tokens. The sub-group is selected using the peers' reputation value. These peers are called ***trusted peers***. From the trusted peers a quorum of peers will be selected to create a signature with the system's shared private key using threshold cryptography. Thus, a (t, T)-threshold scheme is applied, where the threshold t determines the required quorum size and T denotes the total number of trusted peers. Each trusted peer has to receive a share of the private system key.

Tokens contain a basic set of information. The owner's identification enables it to clearly attribute a token to a peer. A unique token identification allows the detection of double spending. A signature with the scheme's private key proves the tokens' validity, ensure its integrity, and prevent forgery.

Due to these security features, it is possible to store tokens locally at the owner peers. Furthermore, tokens cannot be stolen by other peers.

Usage of Tokens

Peers will use tokens in transactions for consuming resources and services. Then the token will become a transaction receipt. The consuming peer (consumer) will add accounting information, sign it with its private key, and hand over the token to the transaction partner (resource or service provider). Spent tokens cannot be stolen, because the receiving peer's information is stored in it. As a basic mechanism for coordinating consumption of resources and services in the system, service providers can swap such received tokens for new tokens. The new tokens are created according to the process for issuing tokens. This overall process will be called ***Token Aggregation.***

Furthermore, such received tokens will be called ***foreign tokens***, in contrast to ***own tokens***, which have not been used in a transaction. Token aggregation swaps foreign tokens for own tokens. Hence, tokens are transient objects and represent the system status.

When tokens are used, there must be a mechanism that detects potential double spending of tokens.

Double Spending Detection for Tokens

In order to detect double spending, there is a distributed hash table (DHT) built by all peers in the system. As tokens are issued to specific peers, for each peer there is a remote account in the DHT where information is stored regarding which tokens the peer spent. These remote accounts are called aggregation accounts. Aggregation accounts have to be replicated in the DHT in order to avoid information loss due to churn of peers. Furthermore, a peer should not be able to modify information in its own aggregation account that would defraud the system, e.g., a peer should not be able to remove tokens from the spent-tokens-list in order to double spend.

Accounting Information

Accounting information about a transaction is stored in foreign tokens at the providing peer. At the requesting peer, accounting information can be stored through copies of spent own tokens. This information can be verified by the information stored in the requesting peer's aggregation account, as well as by querying the providing peer.

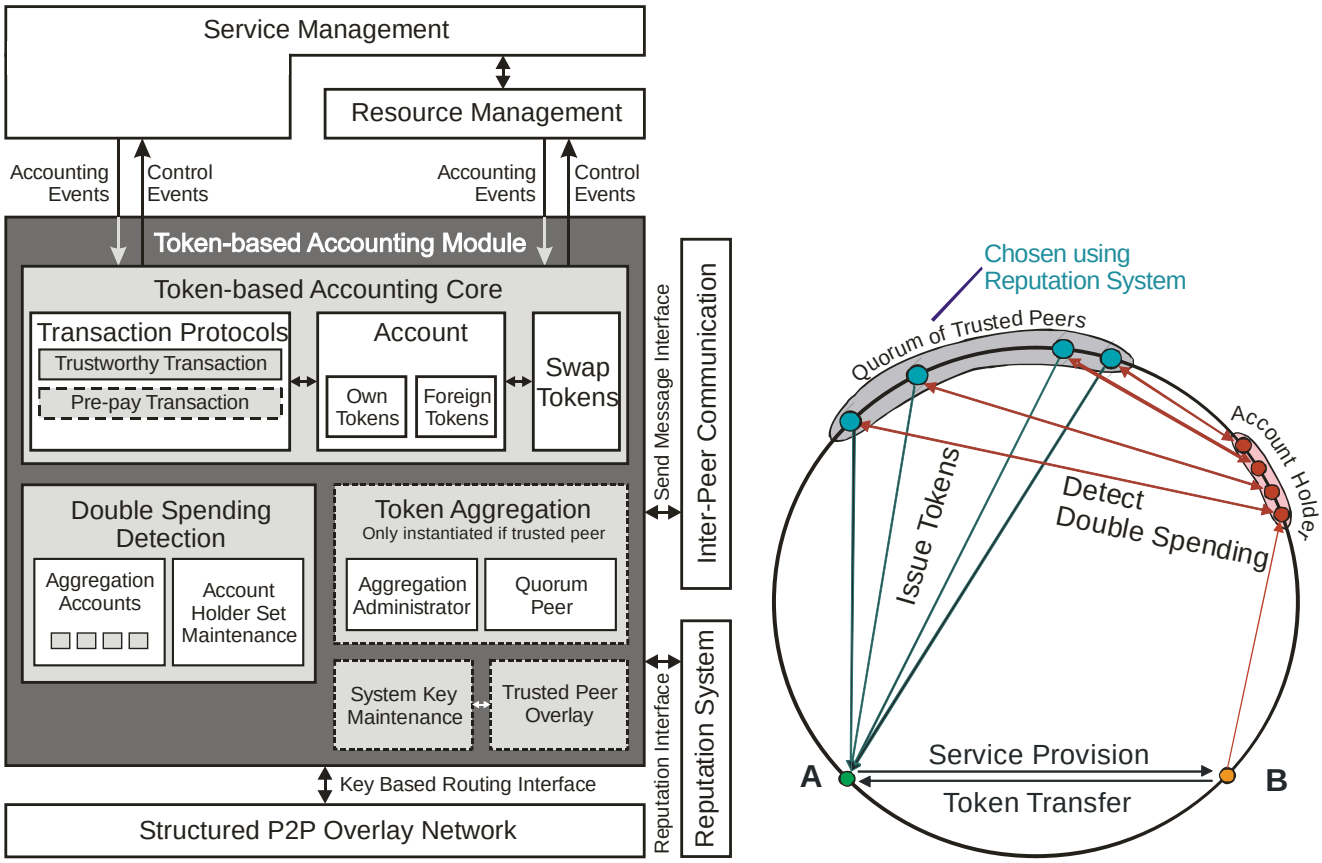
Token-based Accounting Architecture

The token-based accounting framework translates to the architecture depicted in Figure 3.6 a. Service and resource management generate accounting events and send them to the token-based accounting module. This can also send control messages to service and resource management; for example, if a service provider does not receive further tokens from the receiver, the service delivery can be stopped.

The three basic architecture components, Token-based Accounting Core, Check for Double Spending, and Token Aggregation do not interact within one peer but only with remote peers using the Send Message Interface. The inter-peer communication is depicted in Figure 3.6 b. All architecture components can directly communicate with the reputation system, which is beyond the scope of this dissertation. Furthermore, the token-based accounting module has an interface to the structured p2p overlay network that connects the peers. It is used to find peers, e.g. in order to send a message to an account holder set.

The token-based accounting core component implements the account for own and foreign tokens, the transaction protocols, as well the required functionality to swap tokens. Transaction protocol as well as swap tokens both access the account for storing and retrieving tokens. The Double Spending Detection component implements the aggregation accounts a peer is responsible for, as well as the aggregation account maintenance mechanisms. The Token Aggregation component is only instantiated at trusted peers. It has its own interface to the trusted peer overlay used for maintenance mechanisms for the system trusted peer, such as proactive threshold mechanisms. Both components, System Key Maintenance and interface to the trusted peer overlay network are also only instantiated at trusted peers.

In conclusion, the token-based accounting scheme consists of four building blocks: the three basic protocols *Token Aggregation*, *Double Spending Detection*, and *Transaction*. Additionally, a core part of the token-based accounting scheme is the *Token Structure*, which is the first of these four building blocks



(a) Building blocks

(b) Functional overview

Figure 3.6: Token-based accounting scheme – architecture overview

discussed. The building blocks mesh closely. Removing one block neutralises the functionality of the whole system. Each of the four blocks are now discussed in more detail.

3.3 Token Structure

In Section 3.2 the core functionality was described, and some of the required information that should be contained in a token was derived. In this section the token functionality is examined in more detail and the final structure of a token is deduced.

3.3.1 Token Functionality

Tokens serve two functions. They act as both a permission object and a transaction receipt. The permission object is issued to a specific peer and when used to consume resources or a service, the token is extended by the inclusion of receipt information. So, it is meaningful to structure the token into these to core parts - permission object and transaction receipt.

Permission Object

The permission object can only be issued directly to its owner. The issuance of a document requires a valid signature. Furthermore, a permission object should be restricted to a one time only usage. Accordingly, forgery and double spending must be at least detectable. If forgery and double spending can be detected at the time such an object is used, forgery and double spending could even be prevented. This is the preferred result. Detection of double spending requires that a token can clearly be identified. Protection from forgery requires that the signature validating a token is strong.

Transaction Receipt

A transaction receipt requires all relevant information about a transaction, include the transaction partners, the transaction object, and further information relevant about the delivery process of the transaction. All transaction relevant information must be authenticated and its integrity and non-repudiation must be ensured. This can be done by a signature of the peer generating the transaction receipt information.

Peer Identification vs. Anonymity

The functionality of tokens is somewhat similar to the functionality of a virtual currency. An important characteristic of currency is its anonymity. However, tokens are used primarily as issued receipts and not as a virtual currency. Thus, tokens must not have the characteristics of virtual currency, namely anonymity and untraceability (CFN90). Therefore, tokens have a clear owner that is contained in the token. However, using another layer of encryption like in (CFN90), tokens can be made anonymous.

3.3.2 Resulting Token Structure

Figure 3.7 shows the information contained in a token.

New Token Information

A new, unused token contains the top five information fields. The owner id determines the owner of the token. It is the owners public key.

The issuing date and the time in milliseconds, together with the serial number and the owner id, serve as the unique identification of a token. This unique identification is required to enable the detection of double spending, and allows double spending can be traced to the owner.

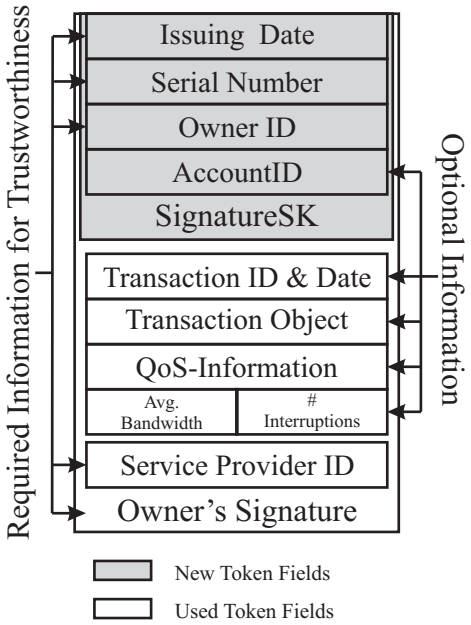


Figure 3.7: Token structure

During the creation of a batch of new tokens, the serial number is randomly selected for every token. Thus, guessing which tokens exist in the system becomes hard.

The account id is used to allocate a token clearly to a specific application. Cross application usage and trade of tokens can be permitted by application providers. The account id field is optional.

The fifth field contains the signature of the information contained in the first four fields, signed with the scheme's private key. This prevents forgery and ensures the information's integrity.

Used Token Information

A used token is basically a receipt, meaning it contains further information about the transaction a token was used for. The service consumer is the token owner.

Before the owner sends the token to the service provider, it adds the service provider's id to the token as well as information about the transaction (such as transaction object, date and information about the quality of the service provisioning). The owner finally signs the complete token using its private key. Subsequently, the contained information cannot be changed by the service provider.

The required information in a token is the information needed for unique identification, i.e., the system signature, the service provider, and the service provider's signature. This prevents tokens from being stolen. Because unused tokens contain the owner, only the owner can spend them. Used tokens are signed and contain the receiver of the token. Only the receiver is allowed to swap used tokens for new, own tokens. A token has no intrinsic value, but rather represents an accounting event. The value of a token is determined in the token aggregation process.

3.4 Basic Transaction Protocol

During transactions, the token-based accounting scheme accounts for resource usage, service usage, or a combination of both. Service usage is valued differently than resource usage. For example, a service detects watermarks in pictures. Since special software is needed to provide such a service, it is valued higher than the sum of the used resources. A token can contain information about the used resources and value information of the service itself. The information is added to a token before it is sent to the service

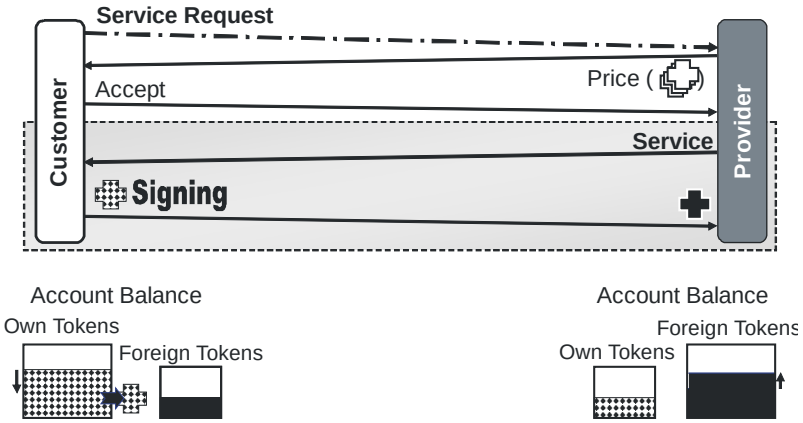


Figure 3.8: Basic transaction procedure

provider. By this means, information contained in a token can be used as basis for external charging and payment mechanisms.

The standard transaction process is shown in Figure 3.8. After a service has been requested by the service consumer *C* , the service provider *P* informs *C* about the terms and conditions of the service, including the number of tokens it expects in return for the service. If *C* accepts the terms and conditions, the service provisioning phase begins.

During this phase tokens can be transmitted before, after, or during the service provisioning. For example a token can be transmitted after 1 MB transferred or after 1 minute service received. Before a token is transmitted, *C* fills in the required accounting information. *C* has no incentive to falsify this information, because it influences only the token aggregation of *P*. Then *C* signs the token with its own private key and sends it to *P*. *P* checks the signature of the received token using *C* 's public key, which can be contained in the token as the owner id or transmitted with the service request. Thus, it can be verified, that the token sender is actually the token owner. In case the accounting data is incorrect, P can choose not to continue to provide the service. Now with each transaction, CS own token balance decreases and PS foreign token balance increases.

Transaction partners could try to gain tokens by not paying tokens after receiving a service or by not delivering the service after receiving tokens. In order to avoid that, transactions can be split into several parts. Then *C* sends a signed token to *P* after *P* delivered a part of the service; for example, *C* sends a token after each MByte data received of a 5 MBytes file transfer.

A further approach that eliminates the incentive for transaction partners to cheat on the other partner is presented in Section 3.7.

3.5 Aggregation Protocol

The Token Aggregation process is used to swap foreign tokens a peer has collected for new tokens issued to that peer. As described in Section 3.2, only trusted peers are allowed to participate in the issuing of new tokens. Trusted peers are selected among all peers according to their reputation value. Trusted peers are in possession of a share of the secret scheme's private key. The *Aggregation Protocol* is built according to the BLS-threshold scheme (BLS04, Bol03) and the URSA-threshold scheme (LKZ+04) (see Section 2.4).

The ten-step Token Aggregation procedure is shown in Figure 3.9.

First the *swapping peer* SP locates a *trusted peer* TP (step 1). SP sends its n collected foreign tokens $(Fn_1, ..., Fn_n)$ to TP using a signed message (step 2). TP checks the foreign tokens for their validity

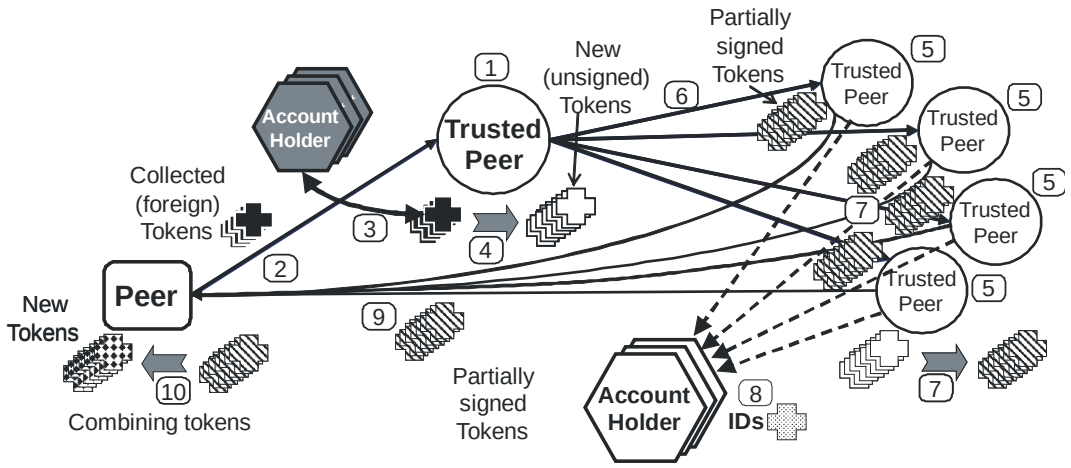


Figure 3.9: Token aggregation

(step 3). Only tokens signed by the owner and spent only once are valid for aggregation. The protocol checking tokens for double spending is described in the following section.

Using the ***aggregation function*** $M = A(Fn_1, ..., Fn_n)$, TP calculates the number M of new tokens SP must receive in return for the foreign tokens. The aggregation function is public and can take any form. TP now creates m fresh, unsigned tokens $(Un_1, ..., Un_m)$ (step 4).

To sign the fresh tokens with the scheme's private key using threshold cryptography (the BLS-scheme (Bol03) or the URSA-scheme (LKZ$^+$04)), TP now locates further trusted peers that build the quorum (step 5). SP is not allowed to choose the quorum of trusted peers itself. This alleviates the problem of potential collusion. The number of required trusted peers to sign a token is determined by the used secret sharing scheme. The system's trustworthiness increases with the size of the quorum of trusted peers.

TP sends the new tokens $(Un_1, ..., Un_m)$ to this quorum of trusted peers (step 6). Each peer of the quorum now checks the validity of the token aggregation process and then signs the tokens with its part of the scheme's private key (step 7). The quorum peers send the identification of the signed tokens to the swapping peer's aggregation account, in order to allow the control of double spending of these tokens (step 8). The resulting partial tokens $(Pn_1, ..., Pn_m)$ are transmitted back to SP (step 9). Finally, SP combines the partial tokens to new complete tokens $(Tn_1, ..., Tn_m)$ (step 10).

In the token-aggregation aggregation protocol there are two possibilities for misbehaviour.

First, the trusted peer TP could create the wrong number of new tokens. In such a case, the swapping peer SP will contact the quorum and send to it a copy of the message it sent to TP in step 2. The quorum peers will validate the number of tokens created and if TP created a wrong number of tokens, the quorum peers will report this to the reputation system.

Second, one or more quorum peers could sign no token or only part of the tokens with its part of the scheme's private key. In this situation SP will send a complaint to TP, which checks SP's aggregation account. Quorum peers that did not sign the tokens and did not send the token identification to the aggregation account can be identified here. TP will contact each quorum peer that did not sign the tokens and requests a justification. If the quorum peer did violate the rules of the token-based accounting scheme, it is reported to the reputation system. Furthermore, if a quorum peer does not respond to the request, it is also reported to the reputation system, as an objective observation for the cause for the misbehaviour cannot be achieved.

Here, a different rating of deviations from protocol is required. Deviations where the cause cannot be determined have to be rated lower than proved violations. This way, trusted peers that fail too often to accomplish their tasks (without proving a deviation from protocol) will be downgraded to normal peers.

If SP did not receive enough new own tokens it will initiate another token aggregation process.

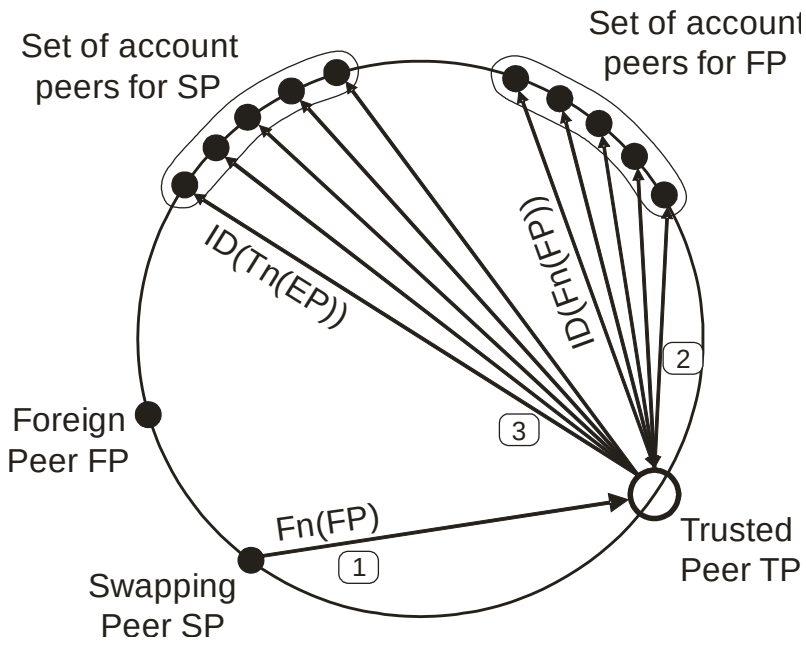


Figure 3.10: Detection of double spending

It is important to mention that the aggregation function adds an additional degree of freedom to the system. With an appropriate aggregation function, specific economic systems can be implemented.

Before the number of new tokens is calculated, the validity of the foreign tokens is checked. Here, it is important to check the tokens for double spending. This process is described in the following section.

3.6 Double Spending Detection Protocol

To check for double spending, a token must be clearly identifiable. To facilitate the check in an efficient manner, for every peer (the account owners) there is an ***aggregation account*** located at a set of account holding peers, called the *account holder set*. The account holder peers are organised in a DHT manner, such as Chord (SMK+01), Pastry (RD01), or Kademlia (MM02) (see Figure 3.10). A requirement for the detailed mechanism is that the applied DHT builds a ring topology (see Section 4.2 for details). If the selected overlay protocol does not provide a ring topology, one has to be added to the protocol.

Account holders hold a list of tokens currently issued to the account owner. The list is filled with the required information during token aggregation. After new tokens have been created (Figure 3.9, step 3), TP sends a list of these new tokens to the exchanging peers account holders (Figure 3.10, step 3).

During the token validity check of the token aggregation process, TP will ask the account holders responsible for a token, if the token is valid (Figure 3.10, step 2). The account holders will remove the token from the list. However, if the token is not in the list, it is an invalid token. TP will discard such a token and the p2p system's reputation mechanism will be informed about the incident.

In order to avoid message manipulation, every message sent to the account holders must be signed with the sender's private key. To keep the list between the account holders consistent, all account holders for one specific account exchange the list whenever the set of account holders changes. This takes place only when peers of that set join or depart from the system. Consistency checks are necessary if the sender does not receive all confirmation messages (for details see Section 4.2). These mechanisms are described in detail in the following chapter.

3.7 Trustworthy Transaction Protocol

Depending on the application scenario, a more trustworthy settlement process might be required, e.g., if tokens are used as a virtual currency. Here, the transaction party that delivers last has an incentive to cheat the other party. It still receives the full benefit, but does not have to deliver its part of the deal. Therefore, a trusted payment procedure was designed that eliminates the incentive to cheat for the transaction partners. In addition, double spending of tokens is not only detectable, but can also be prevented. Figure 3.11 shows the procedure.

After a service request is received, *P* notifies *C* about the terms and conditions of the transaction, including the required number of tokens. *C* answers with the tokens' IDs of the tokens it intends to spend for the transaction. Then *P* contacts the account holders responsible for *C* (*AH(C)*) and checks if the tokens are valid. In the token list *AH(C)* mark these tokens as "*planned to spend*". Using the same tokens in another transaction becomes impossible. If all tokens are valid, *P* informs *C* that the transaction phase can begin. *C* starts the transaction by sending an unsigned token to *P*. *C* loses the token. However, since it is not signed by *C*, *P* cannot swap it for own tokens. *P* has no incentive to not provide the service. Therefore, *P* now provides the agreed service. Because *C* already lost the token, it has no intention keeping the token for itself. *C* will sign the token and send it to *P*. If *C* should fail to send the signed token, *P* can present the unsigned token to *AH(C)*. The possession of the token proves that the transaction had started and the token will be removed from the list and is finally lost for *C*. The aforementioned reputation system provides further incentives against such malicious behaviour. On the other hand, if both peers are consenting to cancel the transaction, *C* does not lose its tokens. *P* informs *AH(C)* in order to remove the *"planned to spend*"-mark in the token list.

Peers that do not fulfil their tasks of a transaction will be reported to the reputation system.

3.8 The Aggregation Function

The aggregation function adds another degree of freedom to the token-based accounting scheme, because it can take any form. In this way, different rules can be realised, e.g., to implement specific economic mechanisms or norms of behaviour.

Simple forms of the aggregation function manipulate only the swapping ratio between foreign tokens and new own tokens. This is meaningful, for example, if the token-based accounting scheme is used as an incentive system. Due to the existence of altruistic peers, some peers might collect a lot of own tokens but will not spent them. If the aggregation function implements a swapping ratio of 1:1, the number of tokens available to peer will drop over time. This would result in a a deflation-like effect, which can reduce the overall system performance. Therefore, here different swapping ratios are appropriate. Obviously, the aggregation function can also be used to control the number of tokens that are available in the system.

Other aggregation functions can take other information into account. For example, the accounting information stored in the tokens could be evaluated. Such an aggregation function could, e.g., give priorities to specific services. For example, in file sharing systems files with high demand or files with very few copies could be prioritised. Tokens can be used as a means to incite users to provide accounted resources and services. Therefore, the aggregation function can also be used to coordinate the resources and services according to the demand. If demand is distinctively higher than supply, an appropriate aggregation function can be used to even out the difference.

Information about peers can also be taken into account. Peers that have a very high reputation value can be rewarded. The opposite is also possible in order to punish malicious behaviour. Another alternative is to give a bonus to the 10% of peers that provided the most service.

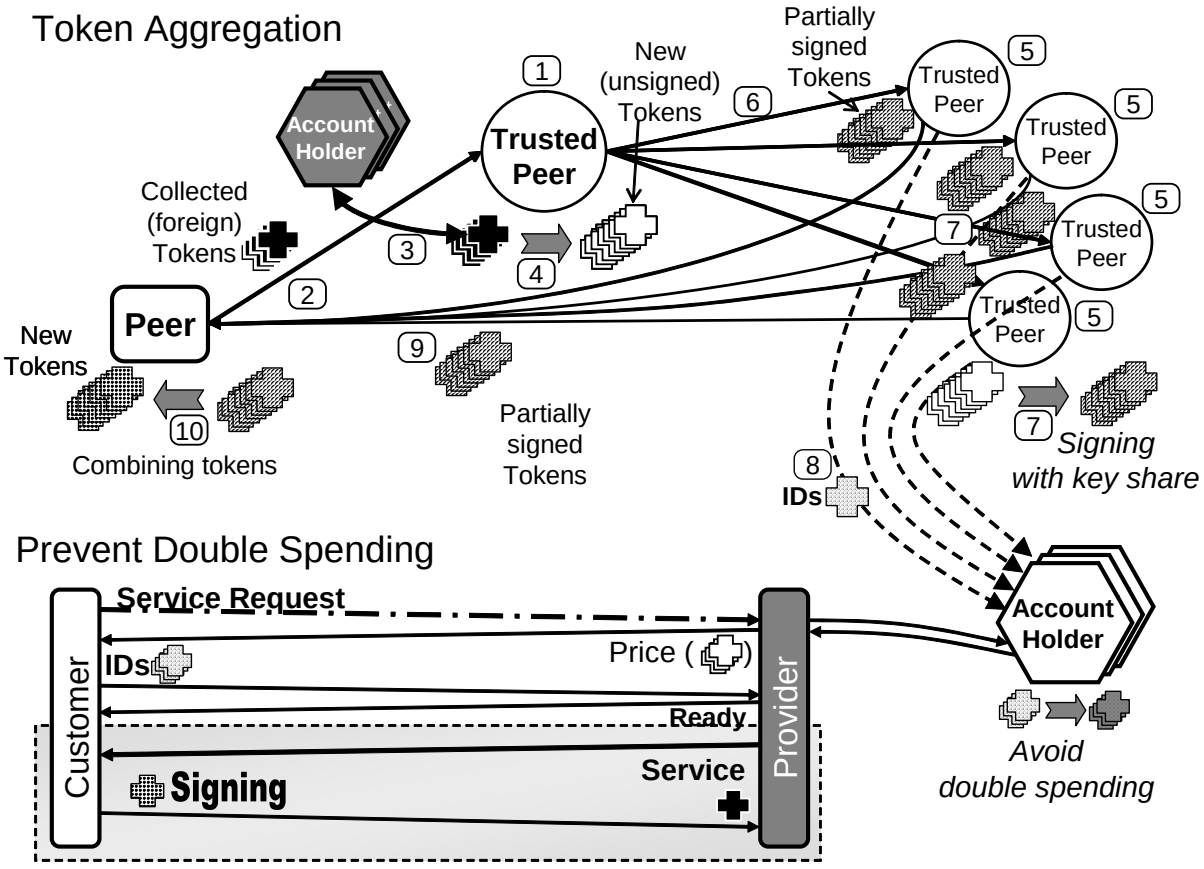


Figure 3.11: Trustworthy transaction procedure

This short discussion shows that there is an unlimited number of alternatives for defining the aggregation function and for implementing different rules, preferences, and economic mechanisms.

3.9 Summary

This chapter presents the system architecture of the token-based accounting scheme. First, the design decisions leading to the system architecture are presented, and then each of the resulting building blocks is described.

The goal of this dissertation is to build a p2p accounting system with intrinsic cooperation control. Based on the requirements and design goals stated in Chapter 1, the system framework is deduced step-by-step. For each design step, the design alternatives are evaluated.

Accounting requires receipts for transactions that were performed by the system's peers. Decentralised cooperation control requires permission objects; peers are permitted to receive accounted resources and services only if they are in possession of permission objects. The derived solution combines the transaction receipts and permission objects into one object called a token. Tokens are issued to peers using a fully decentralised process. Peers spend tokens for consuming accounted resources or services. Furthermore, tokens are allowed to be used only once. When a peer runs out of tokens, it cannot consume any further resources or services. A peer can request additional tokens by swapping foreign tokens it received for providing resources or services for new tokens. The system-wide aggregation function computes the number of new tokens a peer will receive. This combination implements effective decentralised cooperation control based on accounting data.

The aggregation function adds another degree of freedom to the token-based accounting scheme and can take any form. In order to implement different rules and policies in the p2p system, it can, for example, consider accounting information, reputation values, etc.

The system architecture that results from this framework consists of four building blocks: token structure, transaction protocols, token aggregation, and detection of double spending. The token structure ensures information integrity so that tokens cannot be forged or stolen. Therefore, in order to issue it, it contains a signature with a system-wide private key. The transaction protocols define the process how peers "spent" tokens in order to receive accounted resources or services. A trustworthy transaction process was developed that removes the benefits for peers that defraud their transaction partner. Token aggregation is the process that enables peers to swap foreign tokens they received for providing resources or services for new own token. In token aggregation using the aggregation function, the number of new tokens is computed that the swapping peer will receive. Then, each token is signed with the scheme's private key by a quorum of trusted peers using threshold cryptography. Peers might be tempted to spend tokens several times in different transactions. Therefore, detection of double spending maintains for each peer an aggregation account at remote peers. This aggregation account receives information during token aggregation about which tokens are issued to the peer. Also during transactions, the aggregation accounts are updated concerning which tokens have been spent. Thus, double spending can efficiently be detected.

In conclusion, the token-based accounting scheme consists of four building blocks that are closely interlinked. This combination allows trustworthy accounting with intrinsic cooperation control. In the following chapter further details about the protocols are presented. These details are important to demonstrate the viability of the token-based accounting scheme, as they further enhance the scheme's trustworthiness.

4 Relevant Details About Protocols

Chapter 3 described the token-based accounting scheme's architecture. Understanding the basic processes, there are still several open issues and new challenges arise. This chapter presents the solutions and detailed protocols to these issues and challenges. The focus here is on detecting and eliminating fraud.

Section 4.1 discusses how peers can be clearly identified permanently and how their peer id can be tied to a real world identity.

The usage of account holder sets was introduced in the previous chapter: The account holder set requires a number of mechanisms in order to demonstrate the viability of this concept. These mechanisms are the location of the account holder set, assignment of peers, access to account holder set information, and maintenance of account holder set. The maintenance mechanisms ensure *consistent long-term remote storage of dynamic data in DHT-based p2p overlays*. A special challenge was to find *trustworthy access mechanism to remotely stored data in DHT-based p2p overlays*. These challenges and mechanisms are covered in Section 4.2.

Trusted peers and the aggregation protocol build the basic concept for building a distributed basis of trust in the token-based accounting scheme. Section 4.3 discusses the selection and organisation of trusted peers. The aggregation protocol requires enhancements in order to ensure *rule compliant execution of aggregation tasks in the presence of incentives for defraud*. A new mechanism is presented that allows a random selection of a group of trusted peers where the randomness can be verified by any peer afterwards. This prevents collusion.

In order to guarantee the secrecy of the scheme's private key, proactive secret sharing mechanisms can be applied. Section 4.5 presents the proactive mechanism that can be applied to the two selected threshold cryptography schemes and analyses the traffic created by these mechanisms. Section 4.6 uses these findings in order to analyse different strategies for distributing update shares to the trusted peers. Together with the selection of appropriate threshold cryptography schemes performed in Section 3.2.2, this *builds a distributed basis of trust in distributed autonomous systems*.

Finally, Section 4.7 discusses how failures occurring during transactions can be handled by the transaction partners.

4.1 Peer Identification

For trustworthy accounting, peers must be clearly and permanently identified. This eliminates potential Sybil attacks (Dou02), whitewashing, and allows for the identification of those who try to cheat, collude, or act maliciously. Therefore, as described in Section 3.1, each peer possesses a private/public key pair issued by some certification authority. This key pair serves as basic peer identification. The certification authority is responsible for guaranteeing that peers will not change their identification, and is also maintaining the appropriate controls thereof.

In order to identify peers in a peer-to-peer overlay network the hash value $h = H(PK_i)$ of the peer i public key PK_p is used. For this, it is assume that H is a one-way secure hash function and that there are no collisions for the peer IDs.

4.2 Account Holder Set

The account holder set serves the purpose of enabling an efficient detection of double spending and ensuring that only valid tokens are used in the token-based accounting scheme. Thus, it is an additional security mechanism. In order to achieve this goal an account holder set holds account information for a specific peer; this account information is called a peer's *aggregation account*. The mechanisms designed around the account holder set must achieve two sub-purposes. These are the availability of the information and the correctness of the information.

A certain availability will be achieved using replication of the information on a set of peers. The required number of replicas is determined in Section 5.2.2.

The correctness of the information can again be split into two sub-challenges. They are achieving trustworthiness of the information (avoid its manipulation), and achieving the consistency of the different copies of the information.

The assignment process of peers to account holder sets and the information storage and retrieval mechanism of account information were designed in order to achieve trustworthiness of the information. These two mechanisms are explained in detail. Afterwards the consensus protocol is explained, which is used to achieve consistency of the account information at the different account holders.

4.2.1 Peer Assignment, Information Storage and Retrieval

In order to achieve trustworthiness of information it must be impossible or at least sufficiently hard for an adversary to control a complete account holder set.

There are two mechanisms applied in order to achieve this. First, an adversary should not be able to influence the assignment of the account holders for the account owning peer. This is discussed in Section 4.2.1.2. Second, in order to avoid cheating, e.g., by bribing of account holders by an adversary, an adversary should not be able to detect the position of an account holder set. Thus, the account holders in the token-based accounting scheme should be anonymous. Anonymity in communication networks can be split into sender anonymity and receiver anonymity. Here, receiver anonymity is required. However, sender anonymity has advantages and disadvantages.

If the sender is anonymous, it is hard for both the sender of account information and the account holder to collude in order to manipulate the account information. However, then the account holder set might be a target of denial of service attacks or of false account information as it cannot be traced to the responsible peer. A peer could do that in order to paralyse an account holder set for a specific time. Also, a peer could try to harm another peer by guessing the token IDs of another peer and claiming these have been spent.

If the sender is not anonymous, denial of service attacks or false account information could be traced back to the sender. However, an account holder could contact a sender in order to ask if the sender is willing to defraud the system. This attack requires the collusion of a specific number of account holders, so that the false information will prevail when executing a consensus protocol on the account holder set.

False information is either missing information in order to prevent the early detection of potential double spending or additional information in order to insert forged tokens into system. Double spending will be detected in the token aggregation phase at the latest. Therefore, such an attack will not succeed. For successfully inserting forged tokens into the system it is required that the defrauding peers are able to forge the tokens' system signature.

Accordingly, the described trade-off between the different possible autonomy choices has to be evaluated. In case of sender anonymity, the attacks are initiated by peers requesting information storage in an account. If the sender is not anonymous, the attacks have to be initiated by an account holder. Table 4.1 summarises these potential cheating and collusion attacks.

Table 4.1: Potential Attacks Involving the Account Holder Set

Attacks in presence of sender anonymity	Attacks in absence of sender anonymity
Denial of Service	Collusion in order to inject false information
False account information	
Guessing token IDs, claiming they have been spent	

In conclusion, the attacks in the presence of sender anonymity outweigh the attacks in the absence of sender anonymity. Therefore, for the token-based accounting scheme, a solution should be applied where the account holder set is anonymous, but the sender of messages to an account holder set can be clearly identified.

For anonymity in structured p2p systems different solutions have been proposed, which are reviewed now.

4.2.1.1 Anonymity in Structured P2P Systems – Related Work

A traditional idea how to provide anonymity is mix systems. The idea was first presented in (Cha81). Messages are routed over a multiple of hops, where each hop can determine just the previous hop and the next hop the message travels. Therefore, an intermediate hop can not conclude the sender or receiver. Furthermore, the sender sends the messages encrypted with each hops public key. At each hop one encryption is removed. Therefore, the message appears different at each hop and an adversary cannot observe which way a message is travelling. In (Cha88) an extension is presented that ensures unconditional sender anonymity.

A variation of mixes is called Onion Routing and presented in (RSG98). Onion Routing aims at hiding the sender and the receiver of real-time bidirectional communication and at protection against eavesdropping. Onion Routing uses connections through a sequence of machines called onion routers. Each onion router just knows its predecessor and its successor in a connection. Thus, sender and receiver of data remain anonymous and can not be determined by observing the path a packet travels through the network. Furthermore, data passed along the connection appears different at each hop. This is achieved by encrypting the original message with several layers of encryption. Each packet p is encrypted with the hops h_i public key k_{h_i}. A sender sends the encryption block $\{\{...\{\{p\}_{k_{h_l}}\}_{k_{h_{l-1}}}...\}_{k_{h_2}}\}_{k_{h_1}}$. At each hop, one layer of encryption is removed.

Tarzan (FSCM02) is a peer-to-peer based anonymous network layer. In a structured peer-to-peer system, peers are used as proxies to create a tunnel with its endpoint being a NAT gateway[1] to the IP network. Each hop in the tunnel uses symmetric encryption. Furthermore, the onion routing mechanism is also used. Therefore, data passed along the connection appears different at each hop.

In (HW02) Achord is presented, a variant of Chord (SMK+01) that supports anonymity in a DHT. In Achord it is possible to insert data into the DHT or retrieve data from the DHT without revealing the identity of the inserter or the retriever and that without revealing the identity of the peer storing a document. In order to achieve this, the basic mechanism is tunnelling queries and the query responses over several peers serving as proxies. A peer can not determine if the peer from which it receives a query or query response is the originator of this message.

The AP3 architecture (MOP+04) uses a similar approach to Tarzan for achieving message anonymity. Messages are forwarded over multiple peers to the target peer. The receiver is not able to determine the sender. An anonymous return path is created in order to allow communication with the anonymous sender. Secure anonymous pseudonyms ensure that both sender and receiver are anonymous. AP3 is used for anonymous group communications.

[1] NAT: Network Address Translation. For more details on NATs and firewalls see (Röd02).

In (SL04) Agyaat is presented. It uses unstructured topologies over a structured p2p overlay to ensure anonymity. Agyaat uses groups, clouds, that are connected with one group member to the DHT. A query originates from a group of peers, but it cannot be determined from which peer exactly. The same is true for a query receiver. Only the group identification will be know by other peers, however the exact peer will not be identifiable.

(Cia05) focuses on recipient anonymity, which is much harder to achieve in a DHT then sender anonymity. The presented solution subtracts a random value from the lower bound of its own address interval. This is used to create imprecise fingers. This will not allow an adversary to map the address space of the DHT. However, the author uses a different definition of the chordal ring DHT. A factor C is introduced, called cutoff, which is the largest possible distance of a peer's first finger from it self. It becomes clear, that C is dependent on how dense the DHT ring is populated with peers. This, however, cannot easily be determined exact in a p2p system. Therefore, this approach is difficult to deploy in an actual p2p system. Furthermore, it only hides the exact overlay identification of nodes. However, it does not provide real recipient anonymity.

A Stealth DHT is presented in (BMR+05). Nodes in the network join the DHT, however there are two groups of nodes. Service nodes are normal DHT nodes as known from other DHTs like Pastry (RD01) or Chord (SMK+01). Stealth nodes join the DHT but do not complete the full join process. Therefore, they are invisible to the other nodes in the DHT. Stealth nodes can publish and search objects in the DHT, however they do not receive and forward query messages. Also, they do not store references to objects. As a result there is a super-peer network, but using only one DHT.

After reviewing the related work it remains an open issue how the requirements laid out in the previous section could be met. The basic reason is that all of the presented mechanisms assume that the message sender wants to stay anonymous, and only then can the recipient be anonymous. However, this violates the requirements for the access of aggregation accounts as derived before. Hollick discusses in (Hol04) solutions for dependable routing in cellular and ad hoc networks. However, anonymity is not part of his solution. Thus a new mechanism must be developed. Potential solutions to this challenge are discussed now.

4.2.1.2 Account Assignment

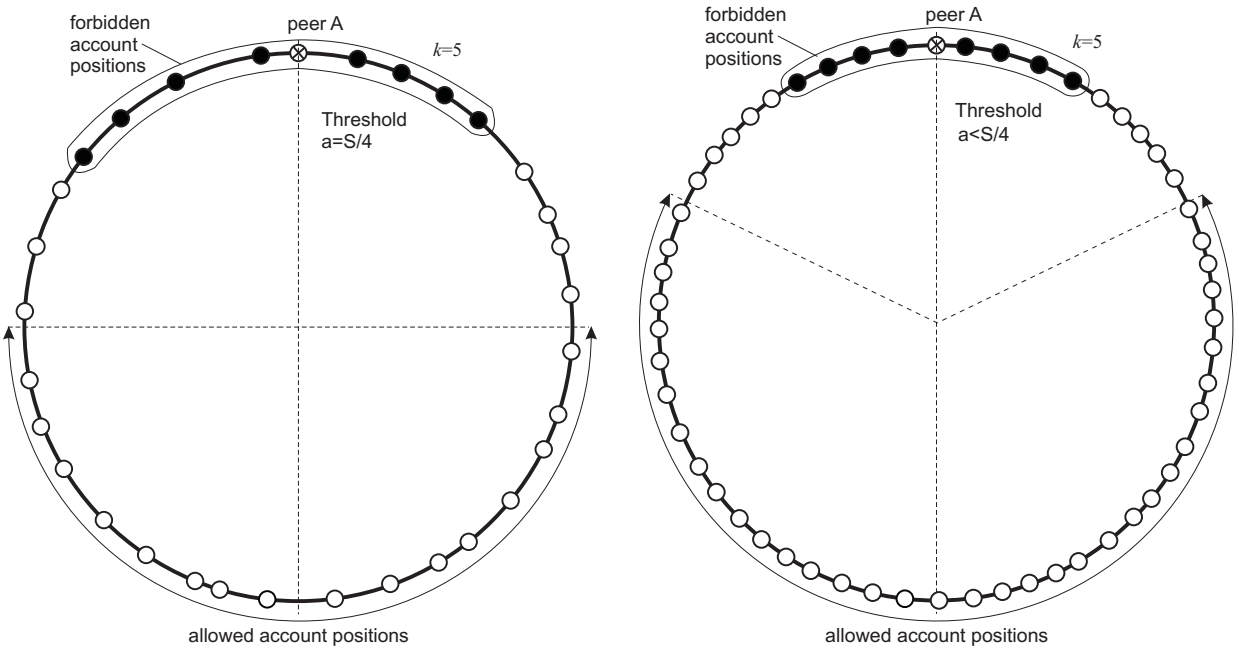
For the token-based accounting scheme, there are two options for which peers could be account holders, either all of the peers or just the trusted peers can be account holders. The advantages of using all peers as account holders are a more evenly distributed traffic load, and a higher number of peers involved when new tokens are created and have to be added to an account. Then, in order to defraud the system a much higher number of peers have to collude. According to the above security discussion, just using trusted peers also has some advantages. In the absence of sender anonymity it is more unlikely that trusted peers will collude in order to inject false account information into the system.

Due to these considerations, the advantages of using all peers as account holders outweigh the disadvantages.

In order to further minimise the possibility of collusion, two further prerequisites should be fulfilled. First, the peers belonging to the same account holder set should be selected randomly. Second, the account holders should not know whose aggregation account they are administering.

The random account holder set selection can be achieved in the token-based accounting scheme by selecting a start peer for an account holder set and then selecting the $k-1$ successor peers in the DHT as further account holders for this aggregation account (k denotes the account holder set size). This process is random, as the secure peer identification scheme presented in Section 4.1 is applied.

In order to avoid that an account holder easily learns whose aggregation account it is administering, an account holder just knows the aggregation account identification (aggregation account ID). The aggregation account ID is calculated using a one-way secure hash function of either the accounts owner's



(a) Small systems (b) large systems

Figure 4.1: Allowed account holder set positions

public key[2] or the account owner's overlay address (which is also derived from the peer's public key and therefore it is static). In order to avoid a peer becoming the account holder for its own account, a system rule will be introduced, that the distance on the DHT between peer ID and aggregation account ID (and between aggregation account ID and peer ID for DHTs with unidirectional distance measurement, like in Chord (SMK+01)) should exceed a specific threshold. This threshold can be system size dependent, that is, it becomes (absolutely) smaller the more peers are present in the system (see Figure 4.1). The threshold a is set to a fraction r of the key space of the DHT S ($a = \frac{S}{r}$). r is chosen in dependency of the system size N to $r = \frac{N}{pk}$, where k denotes the account holder set size and p is a security factor that determines the distance in the DHT between the forbidden account positions and the allowed account positions. How p is chosen depends on the assumption as to how evenly the peer IDs are distributed over the ring. Thus, in a DHT where the nodes are completely evenly distributed, it is $p = \frac{a}{k}$. For a relatively even distribution p is set to 2 and accordingly r is set to $r = \frac{N}{2k}$. In an evenly distributed DHT the threshold would have to be the double size of the potential account holder set involving the account owner.

4.2.1.3 Information Storage and Retrieval

As described above, when storing or retrieving information in or from an account holder set, it is crucial that the actual location of the account holder set is kept secret from the peer accessing the account. However, as indicated in the related work about anonymity in structured p2p systems, the requirements of the token based accounting system are not met by any system discussed there. Therefore, a new solution is required.

The basic goal of a solution sufficient for the problem of keeping the account holder set anonymous is to decouple the aggregation account ID from its actual location in the structured p2p system. However, there must still be a way to find the aggregation account in the structured p2p system.

[2] Here it is crucial that two different one-way secure hash functions are used for the peer identification in the overlay and the account identification

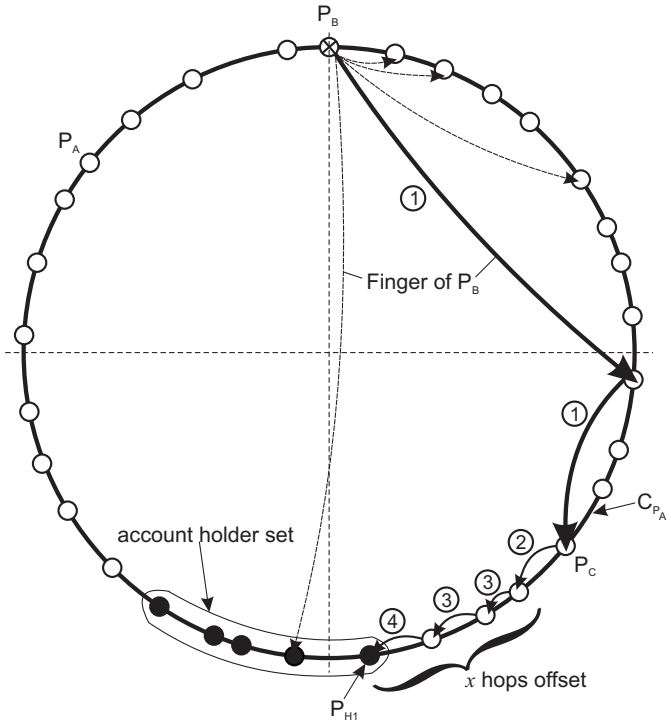


Figure 4.2: Aggregation account position and routing to an account

For the solution to this problem, it is assumed that the p2p overlay used for the account holder peers is a DHT applying a ring topology. In order to decouple an account holder set location from the aggregation account ID, the property of a DHT is used so that the peers are not completely evenly distributed over the DHT ring and therefore, it is hard to predict the exact allocation from key ranges to peers.

Basic Mechanism

The solution that keeps the account holder anonymous from a peer requesting or sending information to it is depicted in Figure 4.2. The first peer of an account holder set is not the peer responsible for the aggregation account ID in the DHT, instead it is the peer following x hops this peer on the DHT. The value x is called the *Account Shift Value*. Accordingly, whenever a peer P_B sends a message the aggregation account of a peer P_A with aggregation account ID AA_{P_A}, the six steps of Algorithm 4.1 are executed:

Algorithm 4.1 Information Storage and Retrieval for Aggregation Accounts

This algorithm is used for storing or retrieving information in an aggregation account given its aggregation account ID AA_{P_A}.

1. P_B sends an aggregation account message to the peer responsible on the DHT for AA_{P_A}. This peer is P_C.
2. P_C receives the message. In its aggregation account query table it stores the query ID. It checks whether this message is a valid message. If not, the message is discarded. If the message is valid, it forwards the message to its successor peer on the DHT, P_D.
3. $P_D$receives the message. It checks, whether this message is a valid message. If so, it checks if it is, itself, an account holder for aggregation account AA_{P_A}. If so, it proceeds to step 5. Otherwise, it forwards the message to its successor peer on the DHT, P_E.
4. Step 3 is repeated until the message reaches the first peer in P_A's account holder set (denoted P_{H1}).
5. P_{H1} processes the message using Algorithm 4.11. If a response is required, the response message is sent to P_{H1}'s predecessor peer on the DHT.
6. A peer receiving a response message checks whether it has the query ID stored in its aggregation account query table. If so, it sends the response back to P_B, using the information stored in the table. Otherwise, it forwards the message to its predecessor on the DHT.

Determination of the Account Shift Value x

The consequence of this mechanism is that a peer cannot determine which peer exactly processes an aggregation account message. It also helps to avoid that a peer trying to access its own account, because multiple of peers have to forward the message to the account. They check whether the account message contains an operation valid in the token-based accounting scheme. If they discover an operation against the rules of the system, they discard the message and send an accusation to the reputation system.

In order to ensure that this mechanism achieves its goal of message receiver anonymity, a peer (or a multiple of peers in cooperation) mapping the DHT must be avoided. Accordingly, strict rules on DHT-related messages must be applied. Achord (HW02) applies several techniques in order to achieve this goal. When a node joins, Achord only allows it a modified version of the recursive method of *find_successor*. During stabilisation only peer n is allowed to call *find_successor(n)*, and here it is only the interactive version. During lookups, a anonymity mechanism does not allow a peer receiving a lookup message to determine the sender. Due to the anonymity and limited knowledge about the network, these mechanisms are also well suited for the handling of aggregation accounts in the token-based accounting scheme.

There are different alternatives for determining the account shift value x.

- **System-wide fixed number**: x is constant for all accounts in the p2p system and never changes. x can be determined using trustworthiness considerations. The more hops a message has to be forwarded to the account holder set, the higher is the probability that there is at least one good peer that will discard messages that breach the system rules.
- **Account specific fixed number**: For each aggregation account i, an own value x_i is determined when the account in created. The account holder set also stores x_i in order to ensure that the account holder set can be located at the correct position in the DHT when the account holder set moves due to peer joins and peer leaves. x_i will be calculated using information stored in

the aggregation account initially, that the account owning peer cannot know (see also Section 4.2.1.5). This information I is hashed and used to calculate x_i using Equation 4.1, where d is the average number of hops an aggregation account position can vary and o is the minimum offset of an aggregation account position from the peer on the DHT responsible for the aggregation account ID.

$$x_i = o + (hash(I) \bmod d) \tag{4.1}$$

In order to initially create an aggregation account at the correct account holder set position, I must be determined in a way that the account owning peer cannot learn it. Therefore, whenever a new peer joins the system and an aggregation account needs to be created, the new peer will contact a trusted peer. The trusted peer first will calculated the peer's aggregation account ID x_i. x_i is then used in the aggregation account creation mechanism, described in Section 4.2.1.5.

- **Account specific variable number**: In contrast to the second alternative, here x_i is variable over time. Thus, it is harder to determine the actual position of an account holder set. However, this also means that the aggregation account has to be moved whenever x_i changes. Therefore, the created traffic overhead is higher. If x_i changes over time, x_i will be recalculated using the actual account information as I in Equation 4.1. A recalculation can take place either periodically or with specific accounting events. Such an event could be a token aggregation of the account owning peer. Each time new own tokens are stored in the account, x_i will be recalculated based on the new account information. The exact information stored in the aggregation account cannot be determined by the account owner. Therefore, it is also not able to calculate x_i. Only account holders are able to do this.

In order to determine the account shift value that x resp. x_i should take, a probability-based solution is applied. The aggregation account shift by value x serves the purpose of x peers checking whether the message sent to the account holder set is an allowed message before the message reaches the account holder set. Accordingly, x can be determined by calculating the probability that the message has to travel over a least one good peer. Therefore, the Hyper-geometric distribution (BGG94) can be applied.

For the application of choosing x the probability $p_{k>0}$ results in $1 - p(0, K, z, Z)$. Furthermore, there must be an assumed specific ratio r of good peers in the p2p system ($r = Z/K$). Assuming there are Z peers in the p2p system, the probability p_A that the message is travelling over at least one good peer if the account holder set position is shifted by x hops results in Equation 4.2.

$$p_A(Z, r, x) = 1 - \frac{\binom{Z \cdot (1-r)}{x}}{\binom{Z}{x}} \tag{4.2}$$

Some results for p_A are listed in Table 4.2.

In order to determine a good account shift value x, the ratio r of good peers in the system must be estimated. There are no studies investigating how many peers in a p2p system act honestly and how many peers do not, because this question has not arisen in p2p systems so far. As the token-based accounting scheme is targeted for serious applications that include commercial applications, it can be assumed that the majority of peers is interested in a trustworthy system. Accordingly, a value for r of 50% is assumed in the following.

Which value of x should be chosen is a trade-off between the achieved trustworthiness of the account holder set lookup mechanism and the created traffic. When x is increased, the average number of hops a lookup message travels is also increased. However, the complexity of a lookup operation remains at $O(\log N)$.

Table 4.2: Required Hops for Aggregation Account Shift for $Z = 10000$

p_a	r	x	p_a	r	x	p_a	r	x
0,699281447	33%	3	0,875037504	50%	3	0,66	66%	1
0,865053996	33%	5	0,937537502	50%	4	0,884422442	66%	2
0,959449229	33%	8	0,968781248	50%	5	0,96071889	66%	3
0,981812011	33%	10	0,984398433	50%	6	0,986652202	66%	4
0,991843863	33%	12	0,9922039	50%	7	0,995465274	66%	5
0,996343223	33%	14	0,99610468	50%	8	0,99845969	66%	6
0,99836082	33%	16	0,9980539	50%	9	0,999476905	66%	7
0,99926537	33%	18	0,999027826	50%	10	0,99982239	66%	8
0,999670826	33%	20	0,999757467	50%	12	0,999939706	66%	9

A peer trying to find out the actual position of an account holder set has to assume that there is a good peer in the lookup chain with a certain probability, and that this good peer will report the attempted cheating to the reputation system. In order to avoid such an attempt it is sufficient to make the probability of discovery sufficiently high so that no peer will try it. Therefore, a probability of larger than 95% should be sufficient. Accordingly, for a ratio of 50% of good peers choosing $5 \leq x \leq 7$ should be sufficient. For a less trustable system with a ratio of only 33% good peers x should be chosen $8 \leq x \leq 12$.

For efficiency reasons a value X_{max} (called *Maximal Account Shift Value*) should be known to the system that is used when a specific account is accessed. X_{max} specifies the maximum number of hops a message should be forwarded when searching an account. If X_{max} is reached before an account is found, it is assumed that the account does not exist. X_{max} should be set to a factor $a > 1$ of the maximum value x can take ($X_{max} = a * x_{max}$). a is chosen greater 1 because an aggregation account can be shifted along the DHT a few hops from its optimal location due to peer joins. For this mechanism to work efficiently (i.e. an account is found if it exists and not too many messages are used if the account does not exist, a value between 2 and 3 are reasonable).

Trustworthiness Discussion

The information storage and retrieval mechanism enables access to aggregation accounts where the receiver is anonymous and the sender can clearly be identified. This is achieved by shifting aggregation accounts a specific number of steps along the DHT-ring away from the peer responsible for the account ID.

The anonymity of the storage and retrieval mechanism is not complete, and the location of an aggregation account can be roughly estimated by other peers. However, an adversary that plans to cheat by bribing or manipulating an account holder set, has either to send messages using the mechanism or to try several peers until it finds the account. In both alternatives, an adversary cannot know if it will send the message to a good peer that will report this to the reputation mechanism. This is the basic concept of a mechanism. As an adversary must be afraid of being detected with a very high probability, it will not start an attack.

One problem with the mechanism is the hop-by-hop forwarding along the ring. Malicious peers could simply not forward these messages. An extension to the storage and retrieval mechanism can be applied in order to detect such behaviour. A peer forwarding the access messages along the ring will send two messages, one message to its first successor, another one to its second successor.[3] If the second successor does not receive a copy of the message from its predecessor, obviously it did not forward the message. This behaviour should be reported to the reputation system and when repetitions occur, the malicious peer should eventually be removed from the system.

[3] In a chord ring, typically each peers stores ten successors.

This mechanism can be applied for all protocols that apply forwarding along the DHT ring, like checking for account existence (see below), etc.

4.2.1.4 Account Holder Set Size

For aggregation accounts, the position and the basic information storage and retrieval mechanisms have been discussed. Still to be determined is the size of an account holder set storing an aggregation account. An aggregation account is replicated over a set of account holders in order to ensure that in presence of churn, the account information is not lost. The account holder set size is an additional security mechanism. A peer is not allowed to access its own aggregation account, but can assume that its aggregation account is available in order to avoid attempts at cheating.

In the related work different solutions have been researched for replica management in p2p systems. On defines in (On04) availability in a redundancy-based system like the account holder set consisting of peer as $Availability = 1 - (p_f)^k$, where p_f is the probability of failure of one peer and k is the number of replicas, which is the account holder set size. Furthermore, On researches two different replica placement strategies, proactive placement and on-the-fly placement. Proactive placement uses a priori determined replica locations. On-the-fly placement uses a peer's online time and uptime probability to decide at which peers to replicate. Unfortunately, in the token-based accounting scheme these mechanisms cannot be applied due to the mechanisms that increase the scheme's trustworthiness.

Knežević researches in (Kne07) a decentralised and self-adaptable replication protocol that is able to guarantee high data availability and consistency in dynamic distributed hash tables. The protocol measures the availability of peers and adapts the number of replicas according to this measurement. In the researched protocol, mechanisms like account handover are not considered. Also, the churn model applied does not conform to measurement results, e.g., like presented in (BQ04, SENB07b, SENB07a) (BQ04, SENB07b, SENB07a). Therefore, it is not clear if this protocol could be successfully transferred to the token-based accounting scheme.

Kangasharju presents in (KRR02) different heuristics for object replication in content distribution networks. However, this problem is not closely related to the account holder set concept. Therefore, these heuristics cannot be applied here.

Knežević also studies in (Kne07) the related work in p2p replication management extensively. He concludes that most approaches assume that peer availability is fairly high, stable, and known in advance. However, this cannot be assumed. Also, due to these assumptions, there is no replication mechanism that conforms with the requirements of the token-based accounting scheme, which arise mainly from the mechanisms for enhancing the scheme's trustworthiness.

Knežević protocol seems to be interesting to adapt it to the token-based accounting scheme. However, for evaluation purposes of the token-based accounting scheme, a specific account holder set size needs to be assumed. Therefore, in Chapter 5, the required account holder set size will be determined in the presence of the mechanisms maintaining the account holder sets, which will be discussed within this chapter. Using a mechanism that adapts the account holder set size according to churn can be added to the token-based accounting scheme in a further step.

4.2.1.5 Aggregation Account Creation

Having solved the problems where the account holder set should be located, how large the account holder set should be, and how peers can submit and request account information, there are two questions remaining.

1. How is a new account for a new peer created?

2. How can a peer determine if it should belong to a specific account holder set? And as a sub-question: How can a peer determine if it is still the first peer of a specific account holder set?

However, before a new aggregation account will be created, a peer must first detect that this account is missing.

Checking for Account Existence

A peer P_1 (for example a trusted peer) that requires knowledge as to whether the aggregation account for a peer P_C exists will use Algorithm 4.2 (see Appendix D.1.1.1 of details). P_1 sends a *check_aggregation_account*-message to the peer responsible for P_C's aggregation accountID AA_C^{ID}. This message contains a hopcounter, initially set to -1. First, the message receiver checks if it is responsible for this account. If not, it starts forwarding the message along the DHT ring, into both directions (because the account could have shifted behind this peer). Each message receiver increases the hopcounter by one before forwarding the message to its successor on the DHT. The message is forwarded either until a peer receives the message that is responsible for the account, or until the hopcounter reaches its predefined maximum X_{max} (see Section 4.2.1.2). In the latter case it is assumed that the account does not exist. Then, the peer increasing the hopcounter to its maximum will create an *aggregation_account_failure*-message and send it to P_1. Now, P_1 can request the creation of the aggregation account for P_C.

If the aggregation account exists, P_1 will receive an *aggregation_account_status*-message from a peer hosting the account. In order to keep the account location confidential it is important that this message is sender anonymous, i.e. it is a UDP-based message.

Aggregation Account Creation

In order to understand how a new account for a peer is created it must first be understood when this is required and which instance of the p2p system is triggering an account creation.

A new peer can exist in the p2p system without owning an aggregation account. It can offer services and receive foreign tokens in return. Only if a peer wants to have own tokens it requires an aggregation account. Accordingly, the mechanism described in Algorithm 4.3 (see Appendix D.1.1.1 for details) is applied in order to create an aggregation account. The mechanism achieves the goal of a trustworthy account creation, that is, the exact account position will be only be known by the trusted peer requesting the account creation. Figure 4.3 depicts Algorithm 4.3.

In the first step, a peer P_{new}, which does not own an aggregation account yet, requests new tokens. In order to do that the token aggregation protocol is applied. The trusted peer responsible for creating the new tokens, TP_1, will check if P_{new}'s aggregation account exists already using Algorithm 4.2 (step 2). If this account does not exist (step 3), the TP_1 will first calculate the aggregation accountID for P_{new} (see Section 4.2.1.2) (step 3a). Then, TP_1 looks up the peer responsible for this ID in the DHT, P_C (step 3b). The next steps should now determine the exact account position on the DHT, but without compromising the position to the other peers, and without enabling TP_1 to map big parts of the DHT. Therefore, the trusted peer has to determine the account position in several steps (step 3f). The exact account position depends on the account shift value x, which TP_1 has to calculate first. Strategies for determining x have been discussed in Section 4.2.1. First, TP_1 will send a *request_peer*-message to $P_C's$ successor on the DHT. P_C's successor will be determined by executing a lookup on the DHT of P_C's peerID plus a small integer value, e.g., 1, in order to get an ID in the successor's ID range. The message requests P_C's y_1-th successor on the DHT, where y_1 is a random value between 0 and x-2. This message contains a hopcounter set to y_i. P_C will decrease the hopcounter by 1 and forward the message to its successor. Each peer receiving the *request_peer*-message will decrease the hopcounter by one and forward the message to its successor. The peer that receives a *request_account*-message with a hopcounter of 0 sends its successor's peerID (peerID of P_k) to TP_1. In order to conceal as much of the account creation process as possible, now TP_1 sends another *request_peer*-message to P_k's successor containing a hopcounter set to y_2. This is done by increasing P_k's peerID by a small value, at least by 1. This is repeated until TP_1 locates the peer that is

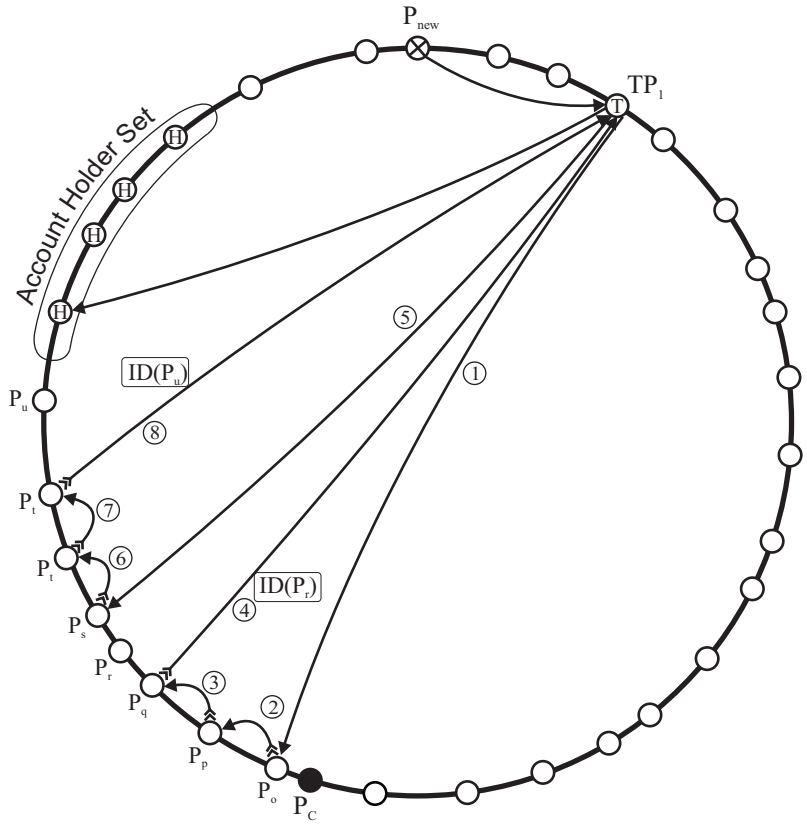


Figure 4.3: Aggregation account creation algorithm

located at x hops distance from P_C. TP_1 has to chose the y-values according to the Equation 4.3, where r is the number of *request_peer*-messages sent.

$$1 + 2r + \sum_{i=1}^{r} y_r = x \tag{4.3}$$

According to Equation 4.3 x is restricted to a specific set of values. Especially for small x, Algorithm 4.3 can be modified in step 3(6)2 that P_i is used as P_j and not P_i's successor.

When TP_1 has located the peer with x hops distance from P_C, TP_1 sends an *init_aggregation_account*-message to it (step 3g). This peer is called P_{AH1}, as it will be the first account holder in the account holder set of that account. This message contains the aggregation account ID of the aggregation account to be created. After creating the account using Algorithm 4.4, P_{AH1} sends an *aggregation_account_confirmed*-message to TP_1, confirming that the account was created (step 3h). In the last step of the algorithm, P_{new}'s account is ready to be used and TP_1 will create the requested tokens for P_{new} and store the corresponding information in the aggregation account using Algorithm 4.11.

When a new aggregation account is to be created, the account has to be initialised after the position of the account was determined. The aggregation account initialisation describes Algorithm 4.4 (see Appendix D.1.1.1 of details). After the first peer of the future account holder set has received the *init_aggregation_account*-message containing the aggregation account ID (step 1), it has to form the account holder set. Therefore, it creates an *create_aggregation_account*-message and forwards it to its successor on the DHT (step 2). This message contains the aggregation account ID, its peerID, and a hopcounter set to 1. The hopcounter has the job of keeping track of the account holder set size. Furthermore the peer remembers the hopcounter position as its position within the account holder set. This is required for Algorithm 4.6. The peerID is the first entry in a list that will contain all account holders at the end of step 4. A peer receiving the *create_aggregation_account*-message will add its peerID to the list and

increase the hopcounter by one. It remembers the hopcounter's value as its own position within the account holder set. Then it will forward the message to its successor on the DHT (step 4). This is repeated until the hopcounter reaches the preferred account holder set size (see Section 5.2.2) or until the message has traversed the complete DHT ring. The latter situation happens only in very small systems. This peer is the last peer in the account holder set. It will create an *aggregation_account_complete*-message. This message announces the membership in account holder set to all account holders. It contains the list of peerIDs from the *create_aggregation_account*-message. The message is sent back hop by hop over all account holders to the first account holder P_{AH1} (step 5). Each peer instantiates a new aggregation account and stores the list of peerIDs as current account holder set in the account. When P_{AH1} receives the message it additionally creates an *aggregation_account_complete*-message for the trusted peer that initiated the account creation (step 6).

Trustworthiness Discussion

The account creation mechanism uses a multi-lookup approach executed by a trusted peer in order to approach the account position. This way no normal peer learns the exact account position.

During the account creation process a trusted peer learns the position of the aggregation account created. Also, it gains partial knowledge about the area of the DHT-ring where the aggregation account is located. An adversarial trusted peer could exploit this knowledge in order to directly address an aggregation account. However, the exact position of the account in the DHT-ring changes as soon as a peer joins or leaves between the account ID and the aggregation account position. Furthermore, each account holder will check if messages sent to the aggregation account are valid messages or not. As the account holders are selected randomly, an adversarial trusted peer has to assume that its attack will be reported to the reputation system. This aggravates all attempts at forming a coalition of colluding peers. Also, at account creation time it is not interesting to try to take control over an aggregation account, because it does not contain a lot of information. If an adversarial trusted peer generates an account at a wrong position, account management will repair that. As a result, there are no serious trust issues during an account creation process.

4.2.2 Account Holder Set Maintenance

Whenever a peer joins or leaves the p2p system account holder sets may be affected and aggregation accounts might have to be moved. There are four different cases that might occur:

- A new peer joins the system at a position in the DHT between the peer responsible for an aggregation account ID and the account holder set. Accordingly, the distance of the account holder set to the peer responsible for the aggregation account ID increases.
- A new peer joins the system at a position in the DHT between the first and the last peer of an account holder set. Accordingly, the account holder set is not a continuous chain of peers anymore.
- A peer leaves that is located in the DHT between the peer responsible for an aggregation account ID and the account holder set. Accordingly, the distance of the account holder set to the peer responsible for the aggregation account ID decreases.
- An account holder leaves the system. Accordingly the aggregation account is one replica short.

All these cases need to be handled. Obviously, the latter case is the most critical one. If such events accumulate, an aggregation account might be lost. In order to handle these cases, the aggregation account management can be structured in three algorithms: Detection of replica numbers, detection of correct account holder set position, and account movement. Keeping the replicas consistent is discussed in Section 4.2.4.

4.2.2.1 Detection of Replica Number

Before any of the other two algorithms are executed, an account holder set has to determine if it is still complete, i.e. if an account holder left the system without a handover of the aggregation account to another peer. In order to check if the account holder set is still complete, Algorithm 4.5 is applied. The algorithm can be initiated by any account holder. In order to keep track of an aggregation account, the algorithm needs to be executed periodically. Thus, any account holder that did not receive a *detect_AHSS*-message for a specific time span will initiate Algorithm 4.5. The algorithm uses reliable authenticated connections, as all account holders must know each other and anonymity is not desired within an account holder set.

As explained in Algorithm 4.4, each account holder holds a list of all current account holders in the set. The account holder initialising the algorithm sends a message to all account holders it currently knows, containing the list of current account holders. Each peer receiving this message will update the received list according to its current knowledge and replies with an updated list. Finally, the initialising account holder composes the actual list of account holders according to the received responses, distributes this to the other current account holders, and informs peers which are not an account holder anymore. If the number of current account holders is below the target account holder set size, it initialises adding an account holder or the detection of the correct account position algorithm (Algorithm 4.6).

For details about this algorithm see Appendix D.1.1.2.

Trustworthiness Discussion

The mechanism to detect the current account holder set size uses one-to-all communication within the set in order to count how many peers respond. In this mechanism, there is no possibility for a cheating peer to influence the resulting account holder set size. If the account holder administering this mechanism is malicious, it could send a wrong table of actual account holders to set. However, any account holder is free to check if the old account holders still exists. If this is performed on a random basis a malicious peer could be detected and reported to the reputation system. Furthermore, the overhead is kept low.

Account holders could also try to free-ride and not respond to any access requests. However, it could still respond to the *detect_AHSS*-Algorithm in order to hide its free-riding. Such free-riding is best detected by the first account holder, because it detects account holders different response behaviour to different message types. Free-riding peers will be reported to the reputation system.

In conclusion, the *detect_AHSS*-Algorithm does not raise any trust issues.

4.2.2.2 Detection of Correct Account Holder Set Position

Each account holder set must check periodically whether the aggregation account is still located at the correct position in the DHT. The position changes through joins and leaves of peers that are located between the peer responsible for the aggregation account's account ID and the actual account position (which is determined by the account shift value x).

In order to determine if an account needs to be moved, Algorithm 4.6 is applied. The first account holder P_{AH_1} of an aggregation account send a message containing a random value to the peer that is responsible for the aggregation account's accountID in the DHT. The random value serves as hopcounter and P_{AH_1} remembers the value. The peer receiving the message increases the hopcounter by one and forwards the message along the DHT ring. Each peer receiving the message will do the same. When P_{AH_1} receives the message again, it can compute the current account offset using the current hopcounter value of the message and the initial hopcounter value. If the result differs from the account shift value x, then P_{AH_1} initialises Algorithm 4.8 for moving the aggregation account.

For details about the algorithm see Appendix D.1.1.3.

Trustworthiness Discussion

This mechanism determines the current account offset uses a random counter in order to count the hops between the peer responsible in the DHT-ring for the account holders account ID and the current first holder. This mechanism does not introduce any trust issues, because manipulations of the hop-counter by malicious peers would be repaired when the account is relocated.

4.2.2.3 Account Holder Set Locking

Before an account can actually be moved, it has to be ensured that the account holder set is not processing any *aa_write_requests* in parallel. Otherwise, inconsistencies within the account holder set cannot be avoided. An *aa_write_request* is used to add new accounting information to an aggregation account (see Algorithm 4.11). The challenge to solve is that even in the presence of competing lock requests from different peers, an unanimous result is achieved. In order to achieve this, a variation of the Strawman Protocol presented in (LLZ04) is applied.

First, the peer requesting a lock P_{AH_X} creates a *request_account_lock*-message, which it forwards to all account holders of aggregation account AA^{ID}. All account holders receiving this message will respond with a lock-message to all other account holders in order to confirm the lock request. When an account holder has received more than 50% of the potential lock-messages, it sends a lock-confirm message to the initialising peer P_{AH_X}. When P_{AH_X} received more than 50% of the potential lock-confirm-messages the account is locked and P_{AH_X} can continue with the next action, account movement.

This algorithm has a message complexity of $O(k^2)$; thus, it is desirable to find another, more efficient locking algorithm. However, this is beyond the scope of this thesis.

For further details about the aggregation account locking algorithm see Appendix D.1.1.4.

4.2.2.4 Account Movement

The account movement algorithm is used for two purposes. First, it moves an aggregation account to the correct location on the DHT. Second, it creates a subsequent account holder set with the correct size. In order to achieve this, the algorithm is split into two parts. The first part verifies the location of the first account holder. The second part actually moves the aggregation account to the new position. The algorithms are similar to the account creation (Algorithm 4.6) described in Section 4.2.1.5.

The first part, Algorithm 4.8, is executed by the first account holder of the set, P_{AH_1}. However, the consistency of all account holders must first be ensured. Otherwise, during the account movement existing inconsistencies could increase.

When the new account holders are determined, each new account holder will receive the account data from the corresponding peer in the old account holder set.

Details about the two algorithms are described in Appendix D.1.1.5.

Trustworthiness Discussion

The account-movement algorithm moves an aggregation account from one account holder set to another account holder set. In order to do that, first the new account position is determined using a trusted peer, because otherwise normal peers could learn the exact start position of the account.

A cheating trusted peer could exploit the mechanism for determining a new account position (Algorithm 4.8) for mapping parts of the DHT-ring. Specifically, a trusted peer could start Algorithm 4.8 without the request for this. Repeating this algorithm for the same part of the DHT-ring will finally reveal its complete structure. This would enable it to initialise a collusion attack on specific aggregation accounts. Therefore, it is crucial that peers receiving messages of Algorithm 4.8 repeatedly with in a short period of time, request a confirm for this action from the first account holder.

4.2.2.5 Graceful Aggregation Account Handover

In order to avoid a completely new account assignment when an account holder leaves the p2p system, an account holder should handover the accounts it hosts to another peer. Whenever an account holder plans to leave the system it will execute Algorithm 4.10 in order to do a clean log off. This is called Graceful Handover. In order to do this, the peer leaving the p2p system P_o contacts the last account holder in the set P_k. P_k will send an account handover request to its successors until it finds a peer that is available P_{k+1}. P_k will inform P_o about P_{k+1} and P_o sends its account data to P_{k+1}. Then all account holders are informed about the change in the set.

The peer leaving the p2p system P_o has to perform the graceful handover for all the aggregation accounts it hosts.

Further details about the algorithm are described in Appendix D.1.1.6.

Trustworthiness Discussion

The graceful handover algorithm allows a peer that goes offline to transfer its account information to a new account holder before it leaves the system. This is a slim mechanism in the sense that there is no consistency within the aggregation account achieved. It is assumed that the handing over peer hands over the account truthfully, i.e. does not manipulate the account information. The algorithms presented in the following will achieve consistency for an aggregation account, i.e., it would repair potential false information injected by a cheating peer. As these mechanism will be executed periodically, potential inconsistencies can be tolerated here.

4.2.3 Information Storage

In the previous sections the location, the creation, and the management of an account holder set has been described. Now, it will be explained how account information is stored in an account holder set. This is composed of two parts. These are the process of posting new information into an aggregation account, and the process of querying an account for information. It is also the structure of the information stored in an aggregation account.

4.2.3.1 Storing Information in an Account Holder Set

There are two situations when information is stored at an account holder set. The first situation is a token aggregation, when the information about the new tokens issued to the account owner has to be submitted. The second situation is during a transaction when tokens are transferred. Here, different types of information can be stored.

For both situations, it is important that the information is stored in a trustworthy manner. It must be clear which peer is submitting information. Only this allows for checking whether the message is sent by a peer that is allowed to submit information to the account holder set.[4] Furthermore, the submitted information must be correct, i.e. it must be ensured that it has not been altered. Due to these two reasons, all messages sent to account holder sets must be authenticated and the messages' integrity must be ensured. That is, it is required that the messages are signed by the sender (submitter).

In order to ensure that no conflicts occur within an aggregation account replica, each account holder maintains a FIFO-queue for requesting and posting account information. The FIFO-queue, which is part of the account information, is also transferred with any account movement. Algorithm 4.11 is used for both account information requests and account information posts. The algorithm's first message is

[4] Account owners are under no circumstances allowed to store information in their own account. Other peers are only allowed to store information at the mentioned situations.

chosen according to the purpose of the algorithm execution. The peer P_A requesting information or posting information from /to aggregation account AA^{ID} will create an *a_request*-message and send it to peer P_C, which is responsible for AA^{ID} on the DHT (steps 1- 3). From P_C the message is routed hop by hop along the DHT until an account holder for aggregation account AA^{ID} is reached (step 4). While the message is forwarded, each peer receiving the message will check if it is an allowed action for the querying/posting peer. Peers may not query/post to their own aggregation account. A message for an action that is not permitted will be dropped and a reputation report will be filed about it.

When the *aa_request*-message reaches an account holder, it will create an *enqueue_aa_request*-message from the request message, and forward it to all account holders. This message is identified by the account holder sending the message in a time stamp it adds. This message is used to enqueue it in each account holders event queue. This is required in order to achieve consistency within the account holder set. Thus, each account holder has the same set of events enqueued in its event queue, although not necessarily in the same order. Therefore, a consistent locking of the accounts for account management operations using Algorithm 4.7 is possible.

The account holder receiving the *aa_request*-message first, P_{AH_1}, will distribute the created *enqueue_aa_request*-message to the remaining account holders. Each account holder will enqueue the contained *aa_request* into its event queue (step 5). The event queue is a FIFO queue. When the *aa_request* is processed, a response is generated. The response is either the response to a query or it is a post confirmation.

For a write request, it must be ensured that all or at least the majority of the account holders write the new information into the account. For account queries this is not required. Therefore, *aa_read*_responses are send back directly to peer P_A. However, *aa_write_responses* are sent to P_{AH_1} (step 6).

P_{AH_1} collects all responses. If it receives consistent responses from more than half of all account holders (step 6.b.i), the information was successfully posted into the account holder set. Then, P_{AH_1} will forward the *aa_response*-message to P_A, using an anonymous connection, e.g., UDP (step 6b). However, P_{AH_1} may have received inconsistent messages. If this is the case, it executes Algorithm 4.12.

If P_{AH_1} receives response messages from less than half of the account holders (step 6.b.ii), the information could not be successfully posted into the account holder set. Therefore, it is necessary for the account holder set to be re-established using Algorithms 4.5, 4.6, and 4.8. Therefore, P_{AH_1} creates a *re-enqueue_aa_request*-message. This is used to undo the effects of the *aa_write_request* and execute it again after the account holder set is re-established. Therefore, the *re-enqueue_aa_request*-message locks the account holder set in order to execute an account management round. After the round, the account holders process the *aa_write_request* again and send an *aa_response* (step 8).

The last case that may occur is that P_{AH_1} is receiving response messages from more than half of the account holders, however, less than half of all account holders send consistent messages (step 6.b.iii). That is, there are still enough account holders active, however their accounts are inconsistent. Then P_{AH_1}will also send a *re-enqueue_aa_request*-message to all account holders and execute afterwards the account consistency algorithm, Algorithm 4.12. This will ensure that the *aa_write_request* is reprocessed on a consistent account holder set.

The detailed mechanism is stated in Appendix D.1.2.1.

Trustworthiness Discussion

The algorithm for posting and querying information from an aggregation account extends Algorithm 4.11. Request messages are sent to the peer responsible for the account ID. From there, the message is forwarded along the DHT-ring to the first account holder of the corresponding aggregation account. The forwarding peers as well as the account holders check if the message initiator is allowed to query or post information. There are two situations when information can be posted to a peer's aggregation account. Transaction partners are allowed to post information during the transaction, and trusted peers are allowed to post information during an aggregation process.

Table 4.3: Account Holder Set Information

General Information					
Account ID	Account holder set		Event queue		
Tokens					
Issuing Information		Usage Information			
Date & Time	Serial Number	remarks (intended to spend)	Receiver	Date & Time	Submitter
...	...	...	...	...	...

Table 4.4: Aggregation Information Stored In Aggregation Accounts

Token Aggregations				
Date & Time	Administrator Hash	Administrator Peer ID	Quorum Hash	Quorum Peers IDs

In the trustworthy transaction process, first the tokens intended to spent will be reported to the account holder set. As the tokens' serial numbers are random, the transaction partner can only report these tokens if a trustworthy transaction process is ongoing. During an aggregation process the account holder set first determines the aggregation administrator and afterwards the quorum (the exact mechanism will be described below). The quorum peers also get notified about the aggregation administrator so, quorum peers can determine if the message sender of partial tokens is the aggregation administrator. Furthermore, the account holder set can determine if the quorum reporting the new tokens to it is the selected one.

Posting information to an aggregation account is only possible for allowed situations. Posts in all other situations will be reported to the reputation system. Cheating and collusion can be successfully prevented.

4.2.3.2 Structure of the Aggregation Account Information

The information stored in an aggregation account serves the purpose of administering in a trustworthy manner, which tokens have been issued to the account holder, and which tokens the account holder has sent. Furthermore, some remarks need to be stored, for example, for the Trustworthy Transaction Protocol (3.7), a token must be able to mark as intended to spent.

Thus, Table 4.3 shows the structure of the information stored in an aggregation account. It is split into two parts. The general information is required for managing the aggregation account. The token section stores for each token the issuing information required for clearly identifying a token. The usage information is added or modified with each transaction or token aggregation that the token is used in.

Consistency of the aggregation account information refers to the token section. Each line (i.e. each token) can be observed individually by the consistency mechanism.

In order to enhance the trustworthiness of the issuing information, the quorum and aggregation administrator could also be stored for each aggregation process. If it would be controversial if a token was created according to the aggregation protocol, this information can be used to contact the trusted peers to determine if they took over the stated role in this aggregation process. The required information is stated in Table 4.4. It is important to note, that the account holders fill in this information by themselves. Other peers cannot influence this information.

4.2.4 Consensus Mechanism

The consensus mechanism is responsible for keeping the information stored in one aggregation account consistent over all account holders. In this section, first the prerequisites for the consensus mechanisms are explained, then the related work about consensus protocols is presented, and then the selected protocol is explained. Finally, it is explained when this protocol is to be executed and how this is triggered.

4.2.4.1 Prerequisites and Requirements

The data stored at the account holders is a list of tokens issued to the account owner. In this list, additional information is added each time a token is used. As explained before, each entry contains a time stamp of the peer's local time submitting the entry. Furthermore, it contains the submitters ID.

4.2.4.2 Related Work on Consensus Protocols

The literature distinguishes three kinds of consensus protocols (CDK02).

The first one is the normal consensus. Here all participating processes suggest a value and all of these processes have finally to agree which value is the correct one.

The second type is the Byzantine Generals problem (LSP82), where one process is suggesting a value and all other processes have to agree on that value. Another consensus protocol type that has been suggested recently is the Paxos protocol (Lam01, Lam98). Like in the Byzantine Generals problem, here one process suggests a value and all other participating processes have to agree to this value. The original protocol does not work with Byzantine failures; however it was extended to do this in (Mar05).

The third type is the interactive consistency, where each participating process is suggesting a value. The goal is that all correct processes agree on a vector with the suggested values. Obviously, for keeping an account holder set in the token-based accounting scheme consistent, the normal consensus is required.

Furthermore, different failure models are used. In the normal failure, model processes can fail, however, they cannot act maliciously by adopting a faulty value. The Byzantine failure model is extended to the possibility.

Surprisingly, the most successful consensus protocols in the distributed systems area are not the results of modern research, but have been invented early in the first millennium. Accordingly, these protocols have been named after their ancient archetypes. They are the already mentioned Byzantine consensus protocols and the Paxos consensus protocols. The Byzantine consensus protocol can be further divided in the contributions that build on Practical Byzantine Fault-tolerance, first presented by Castro and Liskov in (CL99), and the contributions that reduce the number of communication steps, which started with the work by Malkhi and Reiter (MR97).

However, there are two major problems when these consensus mechanisms should be applied to p2p systems. First, in (FLP85) it is proved that there is no algorithm that guarantees consensus in an asynchronous system where processes might fail. Second, the theoretic lower bound of required processes is $3f + 1$,where f is the number of faulty processes involved in the consensus mechanism (PSL80). That means for p2p systems, it must be guaranteed that in an account holder set of four peers at most one is faulty; in an account holder set of seven peers at least 2 must be faulty. Depending on the ratio of dishonest peers in the p2p system the probabilities for achieving this requirement are relatively low (see Table 4.5). For larger systems sizes the probabilities of Table 4.5 only change marginally. These probabilities are not sufficient for p2p systems as it must be assumed that the good peers in the system represent not more than 50% of the total peers. Also, here the highest probabilities can be achieved for small account holder sets. This result is contrary to the availability requirement of an account holder set, which improves with larger account holder sets. Furthermore, these mechanisms are relatively expensive, as

Table 4.5: Probabilities for Byzantine Consensus Compliant Account Holder Sets for System Size of 1000 Peers

	Percentage of Good Peers			
Account Holder Set Size	33,33%	50%	66,67%	80%
4	11,04%	31,21%	59,32%	82,10%
7	4,45%	22,57%	57,14%	85,44%
10	1,90%	17,06%	56,02%	88,19%
13	0,84%	13,18%	55,31%	90,40%
16	0,37%	10,32%	54,81%	92,16%
19	0,17%	8,15%	54,43%	93,58%
22	0,08%	6,48%	54,14%	94,73%

their message complexity has typically the complexity of $O(k^2)$, where k is the account holder set size. Accordingly, it is questionable if Byzantine consensus protocols should be applied in the p2p context.

Nevertheless, in (TBLB06) PaxonDHT is presented that builds the Paxos consensus protocol on top of Pastry (RD01) in order to achieve total order agreement. Byzantine failures are not considered. Therefore, in the probabilistic analysis only peer departures are considered, not potential opportunistic or malicious peer behaviour. If the probabilities in Table 4.5 are also considered, the probability for an account holder set that fulfils the requirements for Paxos is too low. Therefore, this approach cannot be used for the token-based accounting scheme.

Due to the lack of a consensus algorithm that matches the requirements of the token-based accounting scheme, a simple majority-based approach was selected that is presented now. This approach can be replaced with an improved algorithm as the research in the field progresses.

4.2.4.3 Account Holder Set Consensus Mechanism

The consensus mechanism creates consensus between the account holders' token sets. One account holder takes the lead for achieving consensus in the complete set. The peer P_{AH_1} first creates an aggregation account summary (step a). It serves the purpose of enabling quick line-by-line comparisons. For each token line, it builds the hash value over the issuing information as line index and another hash value over the usage information as line value. This account summary is sent to all account holders in order to request a consistency check using *request_consistency_check*-messages (step b). The account holders receiving a *request_consistency_check*-message (P_{AH_X}) will also build an account summary (step c). They will compare the account summaries line-by-line. If both account summaries are consistent, peer P_{AH_X} will respond to P_{AH_1} with a *consistent_account*-message. When there inconsistencies P_{AH_X} creates a differences report, *diff*(S_1^{ID}, S_X^{ID}). This report contains the full information of the token line that differs and not the hash values. Otherwise the account holders cannot evaluate which entry is the correct one. The report is sent to all account holders using *request_update*-messages.

Each account holder will collect the *consistent_account*-messages and the *request_update*-messages from all account holders (step d). As this algorithm is only executed either if the account holder set was re-established, or if more than half of all account holders answered to an *aa_write_request*, at least half of all account holders should reply. Also, P_{AH_1} will send a difference report to the other account holders.

If differences exist in the usage information of a token, it must be decided which entries are correct (step e). A line's correct usage information is determined based on simple majority. If an account holder is missing token lines it means that it is missing the existence of this token. In order to avoid peers easily

adding forged tokens to their aggregation account, additional lines are only accepted if at least a specific ratio of account holders p_{nl} of account holders contain already this line. If this is not the case, the line must be deleted (step f).

Each account holder accepts changes to its account based on a local decision. Therefore, after the peers have updated their aggregation account, the consistency algorithm needs to be repeated, until consistency is achieved.

The probability of adding a new line to an aggregation account p_{nl} is a critical system variable. If it is set close to one, the possibility of adding forged tokens to an account is almost eliminated. On the other side, a peer might lose correct tokens, because they are being deleted when only one or two account holders are accidentally missing this line in the account. A good value for p_{nl} is application dependent. Though, as a reference point it should not be smaller than 50% in order to offer higher requirements for adding additional tokens to an account. Also, depending on the preferred account holder size, p_{nl} should be also chosen. That way, that there is at least one peer tolerated that differs in the tokens stored in the account.

The consistency algorithm is critical to the purpose of the account holder set. Therefore, in general account holders that do not follow this algorithm are reported to the reputation system.

For cases when a decision about which token line is the correct is ambiguous (such cases cannot be ruled out), the consistency mechanism can be extended by a call back function. The leading account holder tries to contact the peer stated as receiver of the token and request information as to whether it really owns this token. The message is sent anonymously containing just the account ID as response address. The token receiver is able to respond to the message using an *aa_request*-message.

The exact consistency mechanism is stated in Appendix D.1.3.1.

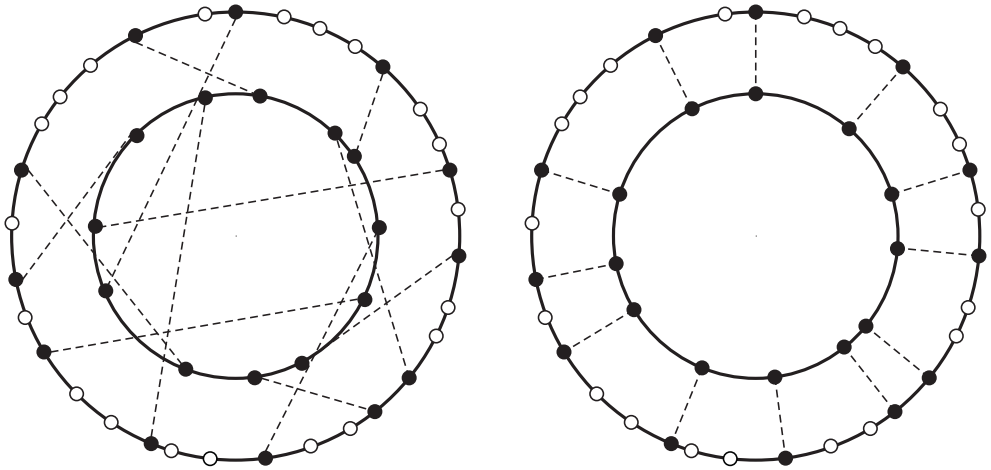
Trustworthiness Discussion

The aggregation account consistency mechanism is executed by one account holder that sends its account summary to all other account holders. The account holders respond with the differences or with a confirm. The consistency is achieved using simple majority.

As simple majority is used to achieve aggregation account consistency the token-based accounting scheme requires that there are at least 3 account holders present when the consistency mechanism is executed. With only two account holders present, a decision about the correct account status is impossible. However, missed transactions can be reconstructed, because aggregation accounts store a history of changes. The history can be verified by contacting transaction partners.

A potential attack of an adversary could be to gain control over a majority of peers in an account holder set. Then it could manipulate this aggregation account. However, this requires knowledge by the adversary about the peers in the account holder set. This is difficult to achieve due to the access mechanism that provides a certain degree of anonymity to account holders. Therefore, an attack from inside an account holder set is the most likely scenario. Here, the adversary could change the status information of tokens. If there are inconsistencies between the existing tokens and the aggregation account the correct status can still be accomplished, although this requires effort and time, as it is unlikely that all transaction partners are online when account information is to be reconstructed. Hence, even if such an attack is successful, in order to have impact on the token-based accounting scheme, the existing tokens would also have to mirror the information in the aggregation account.

In conclusion, an attack on the consistency of an aggregation account can be seen as malicious behaviour, as repairing the account information generates high effort. Cheating using the mechanism is only possible in combination with forgery of tokens.



(a) Trusted peers in both overlays (b) Simplified depiction

Figure 4.4: Trusted peer overlay and system overlay

4.3 Trusted Peers

The trusted peers are an important concept of the token-based accounting scheme. In the token-based accounting scheme, the operations requiring a high degree of security and trust are delegated to the trusted peers. Therefore, it is important that the trusted peers are protected against unauthorised intruders. Thus, a peer must earn its place as trusted peer by achieving a high reputation value. In the following the organisation of the trusted peers, the assignment process for a normal peer to become a trusted peer, and the exclusion of untrustworthy peers is discussed. As a conclusion, the incentives for being a trusted peer are briefly described.

4.3.1 Organisation

In the token-based accounting scheme the trusted peers are mainly used for creating and signing new tokens. The shared scheme's private key used for this requires key maintenance, which again requires that the trusted peers are able to exchange messages between them. It is preferable that normal peers do not get involved in this process for security reasons. The less knowledge normal peers have about these activities, the smaller is the risk of misuse of this knowledge.

Thus, trusted peers are organised in their own overlay network. As it is required by the key maintenance algorithms to route to specific peers efficiently, a structured p2p overlay is chosen. A trusted peer is always a member of both overlays, so it can take part in transactions and token aggregations as normal peer. Figure 4.4 a) depicts both overlays, where the black nodes are trusted peers, that are present in both overlays. However in the following, the idealised depiction of Figure 4.4 b) is used in order to explain the concepts that use both overlays. This figure implies that trusted peers have the same order in both overlays, but this is not the case.

It enhances the security of the system if peers cannot simply map trusted peer overlay addresses to normal overlay addresses, because it increases the difficulty of cooperation between trusted peers outside the trusted peer overlay, which is not planned by the token-based accounting scheme. Therefore, the trusted peer overlay uses different overlay IDs, so a trusted peer cannot be reached at the same overlay address in both overlays.

In order to allow that actions in the trusted peer overlay are clearly attributable to trusted peers, secure overlay addresses are also applied here, as described in Section 3.1.

4.3.2 Number of Trusted Peers

For the token-based accounting scheme it would be helpful to determine how many peers should become a trusted peer or which ratio of trusted peers to normal peers is optimal. The optimal ratio could be determined by finding the ratio where the average traffic for a single trusted peer is minimal. The traffic increases with the number of token aggregations the trusted peers is involved in. This number decreases with an increasing number of trusted peers, as the probability for being selected as a quorum peer or aggregating peer decreases. On the other hand, the more trusted peers exist, the more costly the system key maintenance becomes (see Section 4.5).

Determining this ratio is possible, however it is application dependent as the token aggregation behaviour depends on the average frequency peers are involved in transactions and how many tokens are aggregated on average per transaction between the transaction partners. Furthermore, to apply the optimal ratio of trusted peers to normal peers the actual number of peers online in both overlays must be determined.

In structured overlays like Chord (SMK+01), a peer can estimate the number of peers by analysing its successors and fingers, and concluding how densely the Chord ring is populated. The accuracy of this method depends on how evenly the peers are distributed over the ring. In general, it is assumed that random unique identifiers are distributed relatively evenly over the key space. In the token-based accounting scheme, the peer IDs are a hash value of the peers public key. Also, the hash functions generate values that are distributed relatively even over the key space. Therefore, the requirements to apply these methods are met.

A further mechanism for aggregating information in peer-to-peer systems, like determining the number of peers present in the system, was presented by Kempe in (KDG03). This mechanism can be applied in unstructured overlay networks, however it relies on propagating messages through the complete overlay network.

Using the presented alternatives, in the token-based accounting mechanism a trusted peer can estimate the number of normal peers and trusted peers in present in the system, and thus, can decide if additional trusted peers are required.

4.3.3 Assignment

Only peers that have a reputation value above a specific threshold are allowed to be trusted peers. This threshold is reputation system dependent.

For assigning a new peer as trusted peer, first an active trusted peer has to decide which normal peer should become a trusted peer.

The trusted peer to administer this process can be decided, e.g., by a simple rule. For example, the active trusted peer with the highest trusted peer ID should monitor the number of trusted peers in the system. In a structured overlay system like Chord, a peer can decide which peer should be promoted by analysing its successor list. There is no harm to the system if even a small number of trusted peers monitor the number of trusted peers and assign new trusted peers as necessary, because it is not possible to create more than the targeted number of trusted peers. Another alternative to determine the trusted peer with the highest peer ID in the trusted peer overlay is to apply Kempe's mechanism to aggregate information (KDG03).

There are two alternatives for deciding which normal peer should become a trusted peer. If the reputation system allows the searching of peers by reputation, a normal peer with one of the highest reputation values existing in the p2p system is selected. Otherwise, trusted peers learn another peer's reputation value when they engage in transactions with the peer. This way, a trusted peer gets to know a normal peer with very good reputation, it can be assigned as trusted peer.

To become a trusted peer, the existing trusted peer has to first disclose the function that determines a trusted peer's overlay address to the normal peer. Then, the new trusted peer is inserted in the trusted peer overlay. As last step, the new trusted peer must request a part of the shared private key, as described in Section 4.5.2.

The key assignment requires a group decision of t trusted peers, where t is the threshold value of the applied proactive threshold cryptography scheme. Accordingly, at least t trusted peers have to agree that further trusted peers are required. Simply accepting a normal peer in the trusted peers' overlay is not sufficient.

4.3.4 Exclusion

Whenever a trusted peer's reputation falls below the threshold required for being a trusted peer, they must be excluded permanently from the trusted peer overlay. Therefore, it must be stored somewhere that this peer is not allowed to log on to the board overlay anymore. The expelled peer cannot be simply published in the trusted peers' overlay's DHT as the peer would be responsible for storing its own peer ID as sign for exclusion. Therefore, as a simple alternative, the inverted peerID of expelled peers is stored in the trusted peer overlay. For this purpose, the trusted peers' overlay is required to provide redundant caching in order to avoid stored keys getting lost when peers leave the overlay. As an alternative, the peerIDs of expelled trusted peers could also be stored in the account holder sets, where mechanisms for conserving account information exist and could also be applied for expelled trusted peers.

In the trusted peer overlay, each peer periodically looks up all of its connections on the DHT to see if one of these peers has been published on the DHT. Such a peer should be disconnected from the trusted peer overlay as quickly as possible. Therefore, a trusted peer TP discovering that another peer TP_E is to be expelled, closes the connection to this peer. In order to accelerate the exclusion, it also informs TP_E's predecessor on the DHT about the exclusion. TP_E's predecessor typically maintains a finger to TP_E's successor, so that also this peer can close the connection to TP_E. This process is continued, until TP_E is isolated in the trusted peer overlay.

With each forced disconnect of an expelled peer a system key update phase (see Section 4.5.4) is started in order to invalidate the expelled peer's system key share.

When a peer P_r requests a re-connect to the trusted peer overlay, the peer contacted P_c for joining will check if P_r's trusted peerID was published as expelled. If this is the case, P_c will not respond to P_r so that P_r cannot connect to the overlay.

4.3.5 Incentives

Being a trusted peer in the p2p system basically higher traffic for the node. If bandwidth is a scarce resource for a peer, it is probably not willing to take on the additional resource consumption of a trusted peer. Therefore, incentives should be created for being a trusted peer.

A clear incentive within the token-based accounting scheme are tokens. Therefore, for each token aggregation process, the peer aggregating foreign tokens could receive an extra number of tokens. This extra number should be distributed among the peers that participated in creating and signing its new tokens.

Furthermore incentives could be application specific, like better positions in the transaction partners queues or the exclusion from other application specific duties like cashing, replication, or relaying.

4.3.6 Finding Trusted Peers

Several protocols of the token-based accounting scheme require that a normal peer contacts a trusted peer. Normal peers do not have access to the trusted peer overlay for security reasons. Therefore, trusted peers must be found on the system overlay. To do the lookup of a trusted peer efficiently on the system peer overlay an additional functionality is required. Each peer maintains a finger to the next trusted peer on the DHT. This information is maintained with a few simple operations. When a new normal peer P_n joins the system overlay its successor will send P_n the overlay address of the next trusted peer it knows. P_n will store it. When a new trusted peer joins the system overlay, it will send its own overlay address as next trusted peer to its predecessor in the DHT. The predecessor will forward this information backwards along the DHT until the next trusted peer is reached. This can efficiently be done within normal overlay maintenance messages. Leaves of trusted peers are handled similarly. When the trusted peer TP_1 left the system, the peers maintaining a finger to that trusted peer will lookup TP_1's overlay address. The result is TP_1's successor in the DHT, will respond with the overlay address of the trusted peer TP_2 it maintains a finger to. Now the peers can update their finger to TP_2. This is the Locate Trusted Peer Algorithm (Algorithm 4.13) described in Appendix D.2.1.

Depending on the applied DHT, the finger to the next trusted peer will either be an extra finger (e.g., in Chord (SMK+01)) or it also can be one of the normal fingers a peer maintains in more flexible DHTs like Kademlia (MM02).

The Locate Trusted Peer Algorithm is most frequently used during the token aggregation process. Next, an extension to this protocol is described that further enhances its trustworthiness.

4.4 Secure Aggregation Protocol

The token aggregation protocol as presented in Section 3.5 has a security weakness. The trusted peer creating the new tokens can create any number of new tokens. The aggregating peer might bribe the trusted peer in order to get additional tokens. In order to avoid this, a simple solution could be applied. The foreign tokens that are to be aggregated are sent to all quorum peers. They will verify that the number of new tokens created is correct. However, this is expensive in terms of the required traffic, as the complete foreign tokens are required to be sent. The quorum peers are required to check the signatures to validate the tokens. Furthermore, the quorum peers should also check for double spending.

A second approach that produces less traffic uses the organisation of the trusted peers in a separate overlay. The idea is to use a step of indirection to conceal from the aggregating peer the trusted peer that is creating the new tokens. The aggregating peer should not select the trusted peer that is creating the tokens, nor should it be able to influence the selection. Therefore, a process that randomises the selection of the aggregation peer and the quorum is integrated in the protocol.

4.4.1 Protocol Details

The secure token aggregation protocol starts like the normal aggregation protocol. The swapping peer SP sends the tokens it plans to aggregate for new own tokens to a random trusted peer TP_1 of its choices (step 1). However, because SP chooses this peer, it cannot be trusted. For example, SP could have bribed TP_1. Therefore, TP_1 will forward the tokens to another randomly selected trusted peer TP_2. The challenge is, that the random selection must be verifiable in a later step of the protocol. Otherwise, SP could simply select a colluding peer. Thus, in step 2 TP_1 will contact SP's account holder set by sending a *request_account_stamp*-message. Each peer in SP's account holder set will receive the message and will calculate a hash value h_{SP} of the current account information. This hash value is random and its result cannot be influenced, as the content it is created from is fixed for this point of

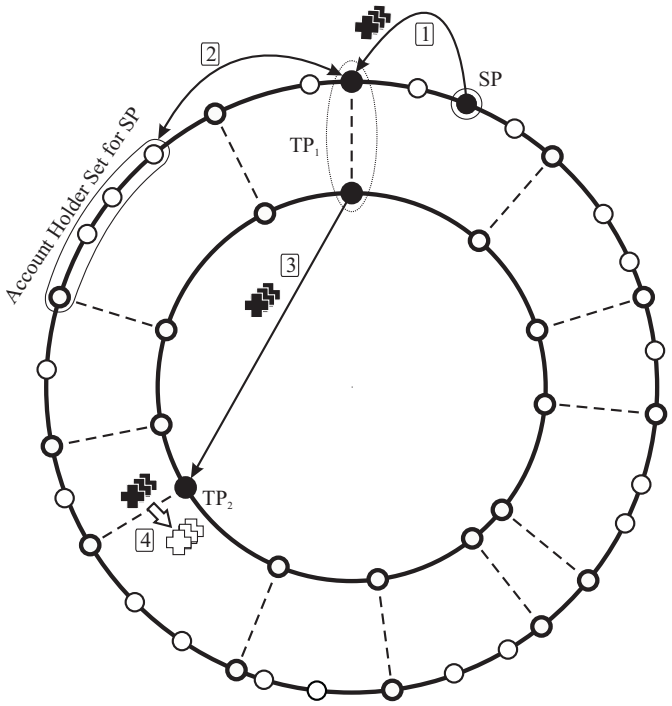


Figure 4.5: Secure token aggregation protocol - part 1

time. Using Algorithm 4.13 described in Appendix D.2.1 the account holder set locates the next trusted peer of h_{SP} in the system overlay. If it is TP_1 (the peer it received the *request_account_stamp*-message from), it inverts the result and repeats Algorithm 4.13. The resulting trusted peer is TP_2. TP_2 is called the *aggregation administrator*. The account holder set is responding to TP_1 with TP_2's peerID.

TP_1 receives TP_2's peerID and forwards SP's aggregation request to TP_2 (step 3). TP_2 will calculate how many tokens SP should receive and create this number of fresh tokens (step 4).

Now, like in the normal token aggregation protocol, TP_2 sends the new tokens' issuing information to SP's account holder set (step 5). The account holders will check if the message was sent by the peer they selected before as TP_2. If not, they will file a reputation report against the sender. Otherwise, the account holders add the tokens to the account with the remark "preliminary", because it will be confirmed later by the quorum that these tokens are correct.

In step 6 TP_2 will send the fresh tokens to the quorum in order to let them partially sign. Equivalent to selecting randomly TP_2, in this step it is also crucial that the quorum be selected randomly. For this purpose a hash of the account information is used. The account information has been updated with the new tokens. The resulting hash value is denoted h'_{SP}. In the response to the *aa-write-request* TP_2 sent to the account holder set in step 5, the account holder set will send a list of t trusted peers to SP_2, where t is the quorum size. The quorum is selected by locating the t next trusted peers of h'_{SP}. The quorum peers are denoted QP_1 to QP_t. In the quorum TP_1 and TP_2 may also take part, as the peers are selected randomly and the quorum as a whole can be trusted.

In step 6 TP_2 sends the fresh tokens to the selected quorum peers. For a system with very high security requirements the responsibility of calculating how many new tokens should be created cannot be assigned to a single peer. In such application scenarios, the foreign tokens are also sent to the quorum peers. The quorum peers can now verify the number of new tokens created. The distribution of the foreign tokens to the quorum peers consumes a lot of bandwidth at TP_2 as the foreign token message is the largest message transferred in the token aggregation protocol and this message must be sent t times by TP_2.

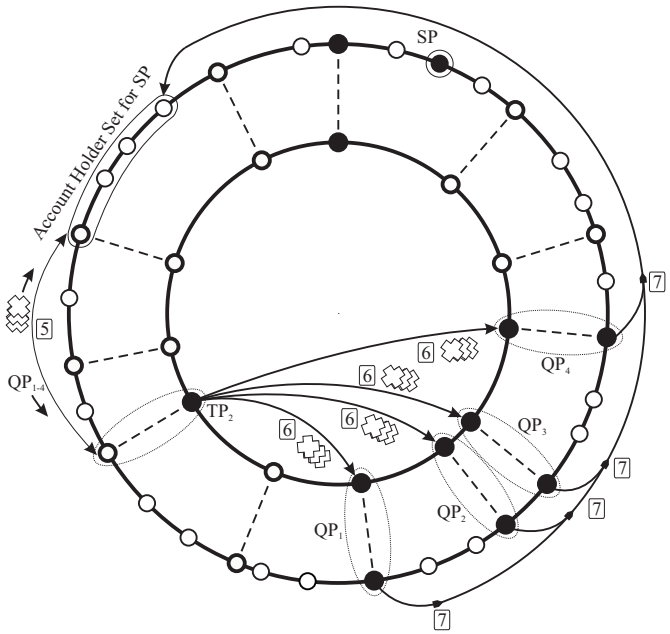


Figure 4.6: Secure token aggregation protocol - part 2

The quorum peers receive the fresh tokens and send the token IDs to SP's account holder set in order to verify the new tokens. The account holder set will only accept the verifications from the trusted peers it selected in step 5.

In step 8 the quorum peers sign the fresh tokens partially and send them to SP. SP will combine the token's signature and create new own tokens.

The detailed algorithm is stated in Appendix D.3.1.

4.4.2 Trustworthiness Discussion

According to the assumption the token-based accounting scheme is built on, Sybil attacks (Dou02) and whitewashing can be successfully prevented. Therefore, this discussion concentrates on the remaining attacks by cheating and collusion. Because tokens are the means for controlling access to accounted resources and services, cheating and collusion have as their aim either the injection of forged tokens into the system or the double spending of tokens.

The secure token aggregation protocol offers the ability to select a random peer as aggregation administrator that will create the new tokens and the ability to select a random quorum that signs the new tokens. The randomness of the aggregation administrator selection can be verified by the quorum. The randomness of the quorum selection can be verified by the account holder set. This has the goal of making it sufficiently hard to create forged tokens.

When all mechanisms for enhancing the trustworthiness of the token-based accounting scheme are applied, there remain two different situations wherein a peer could defraud the system.

- **The cheating peer's account holder set is colluding and the defrauding peer has the knowledge of the scheme's private key.** Here, the cheating peer can itself create tokens. However, in order to inject these into the system, it requires a colluding account holder set that will omit the checks of the sender, and simply add the tokens to the cheating peer's aggregation account.

- **The cheating peer's account holder set colludes and t trusted peers are colluding with cheating peer.** The new tokens are signed by the colluding trusted peers and sent to the account holder

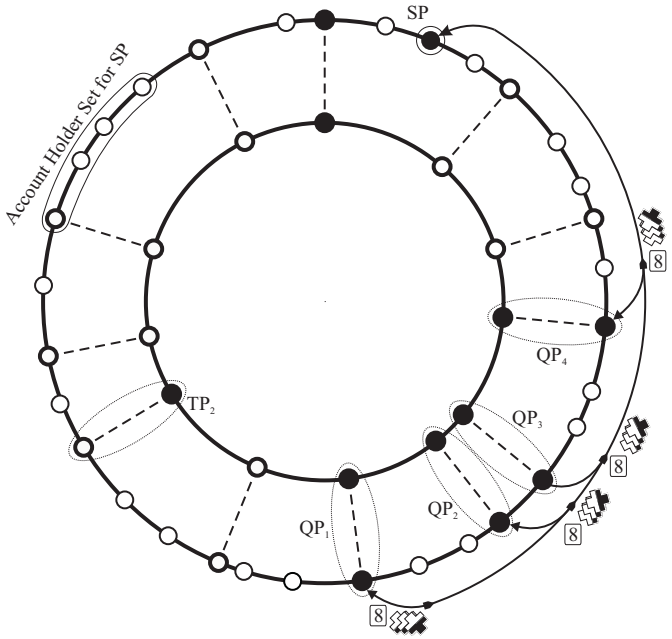


Figure 4.7: Secure token aggregation protocol - part 3

set. The account holder set will omit the checking of the senders, and simply add the new tokens to the cheating peers aggregation account.

In both settings fraud can be detected if the account information is checked frequently and the trusted peers that were supposed to act as aggregation administrator or quorum are not colluding. It is assumed that the account information cannot be manipulated in a way that a hash collision is created that has the aggregation administrator and the quorum as result, which is valid as a secure hash function is assumed.

Section 4.2 explained that a peer is not allowed to contact its account holder set and that the messages sent there are inspected by a number of other peers. Also, the location of the account holder set has a certain degree of anonymity and it changes frequently with joins and leaves of peers. Therefore, it is very hard for a peer to make its account holder set collude in the fraud.

Also, the key management protocols described in the next section aim at avoiding the system key from being compromised.

4.5 System Key Maintenance Protocols

System Key Maintenance is a collection of protocols for initially assigning shares of the scheme's private key to an initial set of peers during bootstrapping (*Initialisation Phase*), creating new key shares for new trusted peers or for trusted peers that have lost their share due to missed updates (*Recovery Phase*), and share updates in order to keep the private system key secure over a longer period of time (*Update Phase*).

In order to keep the scheme's private key secret, all these mechanisms are performed in a separate overlay network. Thus, each trusted peer owns two peer IDs, a normal peer ID and a trusted peer ID in the trusted peer overlay network. The selection of trusted peers was already discussed in Section 4.3.

The phases differ depending on the cryptography scheme used. As described above, either the Threshold BLS-scheme (Bol03) or the URSA-scheme will be applied. As Threshold BLS is the preferred solution, the detailed key maintenance protocols for it will be described here first, and afterwards the adaptations of the URSA scheme will be covered. The protocols are described in detail in the literature and will be discussed here briefly in order to understand the requirements and the resulting traffic.

Table 4.6: Notation for Analytical Comparison of Share Distribution Strategies

Variable	Description	Size [Byte]
MTU	Maximum Transmission Unit on Data Link Layer	1500
r	Length of a modulo	
l	Message information	
v	Traffic volume	
r_{RSA}	Length of RSA modulo	128
$r_{BLS,q}$	Length of BLS modulo q	20
$r_{BLS,p}$	Length of BLS modulo p	64
l_{ID}	Length overlay ID	16
l_{del}	Delimiter. Used in messages to delimit application level information fields	2
l_{TCP}	Length of TCP header	10
l_{IP}	Length of IP header	10
l_{double}	Length of a double value	8
l_{long}	Length of a long value	8
T	Total number of trusted peers	1000
t	Threshold of token-based accounting scheme	17
β	Size of update group	
τ	Number of partitions in tree-based distribution	

In general in this section, the group of peers generating update shares will be denoted as B of size β. A group of trusted peers recovering another peers P_r's share will be denoted D.

4.5.1 Assumptions

For an analytical traffic evaluation of the protocols presented below, it is assumed that all trusted peers own a 1024 bit RSA key pair, that can be used for signing and encrypting message content. BLS is a pure signature scheme and therefore it is inapplicable as a key pair for trusted peers.

For BLS, the typically applied modulo length of modulo $q = 160$bit and modulo $p = 512$bit are used. As message delimiter of message fields 2 Bytes are used.

For the evaluation the messages sent on the application layer are analysed. Furthermore, message fragmentation into IP packets is considered. Below the network layer, the IP packets may travel over many different data link technologies. Due to this heterogeneity, headers from the data link layer are no longer taken into consideration.

For the analysis, it is assumed that TCP and IP are used for the data transfer. In order to take message fragmentation into IP packets into account, an MTU of 1500 Bytes is assumed.

Table 4.6 summarises the notation used for the analysis as well as the remaining assumptions.

4.5.2 Initialisation Phase

In the initialisation phase, each trusted peer P_i receives a share x_i. For both threshold schemes applied, it required that each trusted peer has a unique ID i that serves as the x-value in Shamir's secret sharing that is applied by these schemes. The peer IDs must be in accordance to the applied scheme, i.e. peer IDs must not be larger than the order of the share generator polynomial. For Threshold BLS the peerIDs must be of order $\mathbb{Z}_q$, where q typically is a 160 bit number. For URSA the peer IDs must be of order $\mathbb{Z}_N$, where N is typically 1024 bit or larger. Accordingly, when building the trusted peer overlay, the

key space used when Threshold BLS is applied must not be larger then q. In order to ensure that the key space should be distinctively smaller then q, e.g., for a q of 160 bit a key space of 128 bit should be chosen. For URSA these restrictions are not important as typically DHT-key spaces are distinctively smaller than 1024 bit.

For the initial key generation and assignment, both schemes have in common that a group of peers is required that have to collaborate. For this purpose they require knowledge of the group's peer IDs. For bootstrapping it can be assumed that this group exists; e.g., the application developer provides a group of peers in order to start-up the system. The minimum size of the group depends on the initial threshold t_{inital} of the accounting scheme.

4.5.2.1 Initialisation Phase with Threshold BLS

Threshold BLS-scheme (Bol03) applies (GJKR99) for a distributed generation of the initial shares x_i of the private key x.

The group of peers generating the initial shares x_i first perform a Pedersen-Verifiable Secret Sharing of a random value z_i. The algorithm is described in detail in (GJKR99). Using polynomials of degree $t-1$ the minimum size of the group of peers is t in order to reconstruct the shares x_i.

Each peer P_i selects two random polynomials $f_i(z)$ and $f'_i(z)$ over $\mathbb{Z}_q$ of degree $t-1$ and computes the shares s_{ij} and s'_{ij} for all other peer P_j. Then each P_i distributes these values together with the verification values C_{ik} to all P_j. If there are any complaints two additional broadcasting rounds are required. Each P_i can now compute its shares x_i and x'_i using $x_i = \sum_j s_{ji}$ and $x'_i = \sum_j s'_{ji}$. In order to compute the public key $y = g^x \bmod p$ each player exposes its $y_i = g^{x_i} \bmod p$ using Feldman Verifiable Secret Sharing. Then the public key is computed using $y = \prod_i y_i \bmod p$.

In summary, the initialisation phase for Threshold BLS requires two broadcasting rounds. Assuming a modulus p of length $r_{BLS,\,p} = 512$ bit and an exponent x of length $r_{BLS,\,q} = 160$ bit (as suggested in (STY07)), the messages in the first round have the size of three times 512 bits (plus message delimiting headers fields of length l_{del}). In the second broadcasting round each message has to carry a verification value of 512 bits.

Accordingly, the total upload traffic generated during this phase results in:

$V_{BLS,\,init} = t(t-1) \cdot (2r_{BLS,\,q} + r_{BLS,\,p} + 3l_{del}) + t(t-1)(r_{BLS,\,p} + l_{del})$

This is 3,38 kBytes using a threshold of $t = 17$ and the assumptions in Table 4.6.

Thus, the overall traffic grows with $O(t^2)$. The upload traffic per peer generated results in:

$v_{BLS,\,init} = (t-1)(2r_{BLS,\,q} + r_{BLS,\,p} + 3l_{del}) + (t-1)(r_{BLS,\,p} + l_{del})$

This corresponds to 57.38 kBytes.

Thus, the traffic per peer grows linearly with $O(t)$.

In conclusion, the Threshold BLS initialisation is an efficient way to compute a shared secret private key for the token-based accounting scheme.

4.5.2.2 Initialisation Phase with URSA

URSA is based on RSA signatures. Creating a shared RSA key in a distributed way is very costly compared to generating a shared key in BLS. The reason is that RSA requires a secret pair of prime p and q. Boneh and Franklin presented a mechanism to generate a shared RSA key in (BF01).

The distributed RSA key generation according to (BF01) uses four phases. In the first phase, shared q and p are generated using polynomial secret sharing. Also, q and p are tested with trial divisions if they are prime. This requires a broadcasting round by each P_i for distributing the individual q_{ij} and p_{ij} to all P_j and another broadcasting round to distribute the results of the trial division. This phase is repeated until a q and a p are found that are not divisible by any prime less than some bound B_1.

Using the q_i and p_i from the first phase in the second phase, N is computed using a distributed mechanism that does not reveal knowledge about q_i and p_i. In order to share q_i and p_i, each peer P_i again uses polynomial secret sharing. This requires one broadcasting round and another one for sharing the computed N_i.

In phase three, the found N is tested to understand whether or not it is divisible by small primes in the range $[B_1, B_2]$ for some bound B_2. If this test fails the protocol is restarted from phase one. In this phase, first all peers agree on a random value $g \in \mathbb{Z}_N^*$. Here we assume that one party selects this value in order to avoid another round of polynomial secret sharing. All peers now have to compute a value v_i and share it will all other peers in the group. This is another broadcasting round. v_i is required for the first test if N is a prime. For the second test (a Fermat test) each peer computes a value u_i and shares it with all P_j. This is another broadcasting round.

If the first three phases were successful, in phase four the public exponent is computed in a distributed way, again using a round of polynomial secret sharing and one round of additive secret sharing.

As shown in (ABF+99) the generated traffic is not deterministic due to the distributed guessing of primes. As several broadcasting rounds are used to check if a number is prime, the generated traffic grows exponentially with an increased group size t.

In conclusion, when URSA is used, an application developer should consider generating the system key and the initial set of shares on a trusted server and creating additional shares using the recovery phase.

4.5.3 Recovery Phase

The recovery phase is used to create a new share x_j for a peer P_j using a group of peers of minimum size t. The recovery phase can be used for recovery a lost share of P_j or generating a completely new share for a new trusted peer. This is possible if the threshold scheme applied is based on Shamir's Secret Sharing and not on additive sharing. Both Threshold BLS as well as URSA meet this requirement.

As in p2p systems peers are subject to churn they might miss a share renewal. In order to make the version of a share obvious to peers in all messages of the protocols described below we add a version number.

4.5.3.1 Recovery Phase with Threshold BLS

The recovery phase is used to create a new share x_j for a peer P_j using a group of peers of minimum size t. The recovery phase can be used for recovery a lost share of P_j or generating a completely new share for a new trusted peer. This is possible if the threshold scheme applied is based on Shamir's Secret Sharing and not on additive sharing. Both Threshold BLS as well as URSA meet this requirement.

As in p2p systems, peers are subject to churn meaning they might miss a share renewal. In order to make the version of a share obvious to peers in all messages of the protocols described below, we add a version number.

Simple Recovery

For recovery a share x_r of peer P_r a group of a trusted peers D of minimum size t is required. The simple protocol requires four steps. First each $P_i \in D$ picks a random $(t-1)$-degree polynomial $\delta_i(\cdot) \in \mathbb{Z}_q[z]$ such that $\delta_i(r) = 0$. In step 2, each P_i distributes the individual values $\delta_i(j)$ to all other peers P_j in D. In step 3, each peer P_i now computes its new share of x_r by adding the received values to its share x_i: $x_i' = x_i + \sum_{j \in D} \delta_j(i)$. In the final step 4, all peers P_i send their x_i' to P_r. P_r can now reconstruct its share x_r by interpolating the received x_i'.

Accordingly, for an update group of size t, there are $t(t-1)+t$ messages required to recover one share; each of the messages carries a payload of $r_{BLS,\,q} = 160$ bit plus the version number ($l_{del} + l_{double}$).

Using the assumptions from Table 4.6, this is 0.86 kBytes upload traffic per peer P_i for a threshold $t = 17$, taking IP and TCP headers into account. The complete upload traffic created over all peers results in 14.68 kBytes.

Verifiable Recovery

In the version of the protocol enabling share verification in step 2 each peer also broadcasts the $t-1$ verification values $g^{\delta_{im}}$ together with the shares to the recovery group. The verification values are also sent to P_r. Therefore, the payload of each message sent to the recovery group in step two is $t \cdot r_{BLS,p} + r_{BLS,q} = (t \cdot 512 + 160)$ bit plus formatting headers. The messages sent to P_r have the payload of $t \cdot r_{BLS,p} = t \cdot 512$ bit plus formatting headers. Furthermore, the payload should be signed by P_i with its private key.

In order to verify the received shares and verification values, each peer P_i performs two local tests. If there are any accusations, the accused peers are now excluded from the recovery group.

Step three remains unchanged, except all messages are also signed by the sender. In step four, using the verification values P_r can verify the shares received from the recovery group. It uses only the shares it could validate for reconstruction. Like in the simple recovery P_r reconstructs its share x_r using interpolation.

The upload traffic per peer P_i results in

$$\begin{aligned} v_{BLS,recovery} &= (t-1)(t \cdot r_{BLS,p} + r_{BLS,q} + r_{sig} + (t+2)l_{del} + l_{del} + l_{double}) \\ &\quad +(t \cdot r_{BLS,p} + r_{sig} + (t+1)l_{del} + (r_{BLS,q} + r_{sig} + 2l_{del} + l_{del} + l_{double}) \end{aligned}$$

Using a threshold $t = 17$ and the assumptions from Table 4.6, this is 21.68 kBytes.

Accordingly, the overall upload traffic generated results in

$$\begin{aligned} V_{BLS,recovery} &= t(t-1)(t \cdot r_{BLS,p} + r_{BLS,q} + r_{sig} + (t+2)l_{del} + l_{del} + l_{double}) \\ &\quad +t \cdot (t \cdot r_{BLS,p} + r_{sig} + (t+1)r_{header} + t(r_{BLS,q} + r_{sig} + 2l_{del} + l_{del} + l_{double}) \end{aligned}$$

This corresponds to 368.59 kBytes.

Conclusion

This recovery protocol was not designed for p2p systems but for much smaller systems, i.e. a small set of servers. However, BLS verifiable share recovery is not prohibitively expensive in p2p systems. In (HJJK97) it is assumed that the initial recovery group consists of all system peers that do not need to recover a share. This assumption is not required for p2p systems. The minimum group size is t. In order to have some redundancy the group can be chosen to be larger. The overall traffic grows with $O(t^3)$, the total amount of messages grows with $O(t^2)$. The traffic per peer grows with $O(t^2)$, and the number of messages grows linearly with $O(t)$.

4.5.3.2 Recovery Phase with URSA

Simple Recovery

In URSA, basically the same recovery method (LKZ+04, HJKY95, HJJK97) is applied as is used for Threshold BLS. Here also, the minimum group size of peers P_i recovering a share for peer P_r is t. There is only one difference introduced (KZL+01, LL00). In (HJKY95, HJJK97), recovery polynomials are used with $\delta_i(r) = 0$. URSA uses random values, shuffling factors, instead. This way it can be avoided that P_r can learn any x_i. The t trusted peers in D communicate in advance and exchange shuffling factors d_{ij} in pairs (P_i, P_j). Within one pair, the peer with the higher peer ID treats a shuffling factor as positive, the

peer with the lower peer ID adds the numbers as negative. Each peer in the recovery group D receives $t-1$ shuffling factors d_{ij}. Each peer P_i computes its modified share $x'_i = x_i + \sum_{j=1,j\neq i}^{t} sign(i-j)d_{ij}$, where $sign(x) = 1$ if $x > 0$ and $sign(x) = -1$ if $x < 0$. Then, each P_i sends its x'_i to P_r. P_r can now recover its share x_r using interpolation.

Assuming, the shuffling factors have a maximum size of the RSA-modulus N, the upload traffic generated by this mechanism results in:

$v_{URSA,recovery} = (t-1)(r_{RSA} + l_{del}) + (r_{RSA} + l_{del})$

Using the assumptions from Table 4.6, this is 2.66 kBytes of traffic for a threshold $t = 17$, taking IP and TCP headers into account.

Accordingly, the complete upload traffic created results in:

$V_{URSA,recovery} = t((t-1)(r_{RSA} + l_{del}) + (r_{RSA} + l_{del}))$

Accordingly, this is 45.16 kBytes of traffic.

Verifiable Recovery

In (LL00, KZL+01, LKZ+04) it is stated that either Feldman's Verifiable Secret Sharing (Fel87) or Pedersen's Verifiable Secret Sharing (Ped91b) can be applied. However, this is not true, because:

$g^{s_i} \bmod N = g^{\sum_{j=0}^{t-1} \delta_j (i^j \bmod N)} \bmod N \neq g^{\sum_{j=0}^{t-1} \delta_j i^j} \bmod N$

Possibly, there are ways to circumvent this problem in URSA. However, this is not focus of this dissertation. Accordingly, here URSA is considered without share verification.

Conclusion

The key recovery using URSA creates more traffic than the Threshold BLS alternative. This is due to the larger size of the RSA modulus. The number of messages sent is the same.

Share verification is not yet available for URSA. Accordingly, when doing share recovery it can be meaningful to use groups larger then t for key recovery, in order to add redundancy for malicious peers in the group.

4.5.4 Share Renewal in the Update Phase

The purpose of the update phase is to keep the scheme's private key secret over a longer period of time. In threshold cryptography, adversaries try to get knowledge of at least t shares in order to learn the secret key x. A key update will change all key shares. Therefore, learnt key shares are useless for an adversary after an update phase. Share updates are a basic requirement for keeping a shared key secret over time.

During an update phase, each share of the private key is changed in such a way that the private key and the corresponding public key will not be affected. However, each share x_i will change. The basic idea in order to achieve this is to update the private key's sharing polynomial by adding to it new polynomials δ with $\delta(0) = 0$.

Also, verification values will be updated using $g^{f_k^{new}} = g^{f_k} \cdot \prod_{\forall i \in B} g^{\delta_{ik}} (\mathrm{mod} p)$, where B is the group of trusted peers generating share updates and i are their peer IDs, and k is the index of the parameters of the sharing polynomial $f(x) = s + \sum f_k x^k$ or update polynomial $\delta(x) = \sum \delta_k x^k$.

Due to missed key updates, a trusted peer's share can become outdated rendering it useless for the trusted peer. In this case, a trusted peer can use the recovery phase describe above in order to retrieve again its valid share. In order to make peers aware if they missed a renewal, version numbers are included in the messages.

It is important to note that with an update phase the threshold t of the applied threshold scheme can be increased. Due to the use of polynomial sharing, by adding an update polynomial δ with $\delta(0) = 0$ of degree $(t_{new} - 1) > (t - 1)$ the new sharing polynomial will have degree $(t_{new} - 1)$ and the threshold will increase to t_{new}. This is an elegant way to adjust a low threshold during bootstrapping over time according to number of trusted peers in the system.

The new verification values can be computed, by assuming that the sharing polynomial coefficients f_k for $(t-1) < k < (t_{new}-1)$ were 0 before. Accordingly, the new verification values result in $\prod_{\forall i \in B} g^{\delta_{ik}} (\mathrm{mod} p)$ for $(t-1) < k < (t_{new}-1)$.

Here, the protocols for share renewal are presented. The related literature on proactive secret sharing (e.g. (HJKY95, HJJK97, FGMY97a, FGMY97b, Rab98, SW99, FMY99, CKLS02, NNPV02b, NNPV02a, Bol03, NN04, JSY04, JS05)) typically assumes a small scale scenario with only a few servers that will receive a key renewal. Accordingly, the distribution of the share update is not discussed in here. However, the update distribution strategy is a highly relevant challenge when proactive secret sharing is applied to p2p systems. Therefore, this is discussed in below in Section 4.6

4.5.4.1 Share Renewal with Threshold BLS

Threshold BLS supports a simple share renewal protocol that is secure against passive attackers and a verifiable share renewal protocol that is also secure in the presence of active attackers.

Simple Share Renewal

The share renewal protocol requires a group B of trusted peers of size β to compute and distribute the update shares. This group can have any size. The simple share renewal protocol consists of three steps. In the first step, each peer P_b in B picks a random polynomial δ_b in $\mathbb{Z}_q$ of degree $t-1$ with $\delta_b(0) = 0$. In step 2, each P_i computes $u_{bj} = \delta_i(j) (\mathrm{mod} q)$ for all other peers P_j that should receive an update. Then, each P_b sends these values to all P_j secretly. In the last step, all peers P_i wait to receive all u_{bi} from B. Then they compute their new share $x_i^{(k)} = x_i^{(k-1)} + (u_{1i} + ... + u_{\beta i})$, where k denotes the update phase. Then they erase all variables they used (except of its current share $x_i^{(k)}$).

The traffic generated per peer P_b in B depends on the number of peer to be updated n:

$v_{BLS,\, renewal} = n * (r_{BLS,\, q} + l_{del})$

Total traffic generated also depends on the size β of the group of updating peers B:

$V_{BLS,\, renewal} = \beta \cdot n * (r_{BLS,\, q} + l_{del})$

Verifiable Share Renewal

In verifiable share renewal, each update peer P_b in B also distributes the verification values $\epsilon_{bm} = g^{\delta_{bm}} (\mathrm{mod} p)$ for its update polynomial δ_b together with the update shares u_{bj} (where δ_{bm} are the coefficients of δ_b). Also, each P_b signs the messages with its own private key. When a P_j receives all update shares it verifies them using the verification values. If P_j could verify all shares, it will broadcast an acceptance message. Otherwise it will file an accusation.

Using verification values does not increase the number of messages sent during an update phase. It only increases the message sizes. The traffic generated per peer P_b in B results in

$v_{BLS,\, renewal} = n * (r_{BLS,\, q} + l_{del} + (t-1)(r_{BLS,\, p} + l_{del}) + l_{del})$

Total traffic generated also depends on the size β of the group of updating peers B:

$V_{BLS,\, renewal} = \beta \cdot n * (r_{BLS,\, q} + l_{del} + (t-1)(r_{BLS,\, p} + l_{del}) + l_{del})$

Conclusion

The cryptographic operations within an update phase are the least demanding of all proactive phases. However, the amount of information to distribute is the highest. Accordingly, it must be decided, if each peer P_b within the updating group of peers B should send individual messages to all n peers that should receive an update, or if another method for distributing the information is more efficient. This issue will be discussed in Section 4.6.

Also, in a p2p system it is an open issue how efficiently accusations could be resolved and therefore, if the use of verification values during an update phase will add significant security. This will be addressed in Section 4.5.5.

4.5.4.2 Share Renewal with URSA

Share renewal with URSA uses the same mechanism as Threshold BLS for renewing shares (LKZ+04, HJKY95, HJJK97). The version numbers we introduced for Threshold BLS will also be used here. Furthermore, as described above, verification cannot simply be applied.

The traffic generated per peer P_b in B depends on the number of peer to be updated n:

$v_{URSA\,renewal} = n * (r_{RSA} + l_{del})$

Total traffic generated also depends on the size β of the group of updating peers B:

$V_{URSA,\,renewal} = \beta \cdot n * (r_{RSA} + l_{del})$

Conclusion

Verification of shares for share renewal cannot be yet applied. However, as discussed below, it is questionable if verification of updates is meaningful at all in p2p system. URSA based share renewal generates more traffic than Threshold BLS share renewal due to the larger modulus of RSA.

Also, for URSA the best way of distributing updates, and the most efficient ratio of key updates and key recovery in a p2p system are open issues. Fortunately, for share renewal the message flow is the same as for Threshold BLS share renewal. Therefore, the same distribution strategy can be applied for both proactive threshold schemes.

4.5.5 Handling Accusations in the Update Phase

During system key maintenance accusations are used to detect, advertise, and exclude peers that violate the protocols. When an accusation is filed during any of the system key maintenance protocols, according to (HJJK97) two actions are planned: The first action is not to use the polynomial δ_i and its resulting shares provided by peer P_i. It is important that all peers that plan to use δ_i are informed about the accusation. The second step is to "alert system management" in order to take measures to rectify the misbehaving peer.

As there is no central system management in the token-based accounting scheme, both actions must be merged. This is done via filing a complaint to the reputation system if the verification fails. Before computing the new share, peers have to wait a small period of time in order to allow potential complaints to be filed. Then peers could query the reputation system. This is efficient for share recovery, however in case of share renewal it implies heavy load for the reputation system. Therefore, it depends on the number of trusted peers as well as the reputation system if this method can be applied. If for key update distribution, a tree based distribution strategy is applied complaints could also be distributed with the updates along the tree (see next Section 4.6).

Nonetheless, in a p2p system an intelligent active attacker within the updating peers is hard to identify and to distinguish from malicious peers being updated. An attacker with the group of updating peers B could distribute false shares to only some peers. In this case it is hard to decide if the attacker is the updating peer, or the updated peer that just *claims* it received a false share in order to attack the update process.

Therefore, it could be the best alternative not to actively use accusations to exclude peers and their polynomials from a share renewal process. Accusations are files and are reflected in the peers' reputation value. Peers that could not correctly compute their new share will not try to renew their share by resolving the accusation but will use share recovery instead.

The distribution of the verification values is meaningful despite of not actively resolving accusations. It is not individual information, and could therefore be distributed efficiently. Also, it eliminates some straight forward attacks and also could increase the psychological barrier a user has to cross in order to become an attacker in the share renewal process.

4.6 System Key Update Distribution

For the maintenance of the system key share, renewal is a core concept. The required cryptographic protocols have been described above. Both considered schemes, Threshold BLS and URSA, require the same message flow for a share renewal or a share recovery. When the shares are distributed, each peer requires an individual share. Accordingly, updating large systems is expensive traffic-wise. This section deals with alternative update distribution strategies. In order find an efficient update distribution strategy, there are two basic questions to be asked:

- Should all trusted peers be updated or should only a specific ratio of trusted peers be updated and the remaining trusted peers will recover their share? What is this ratio? In (LL00, LKZ+04) such a scheme is suggested for the URSA update phase.
- Should the group of trusted peers computing the updates send individual messages to all other trusted peers or is it more efficient to build a distribution tree? That implicates additional information included (in particular the peer ID), as well as individual encryption of the update information in order to achieve confidentiality.

When designing an updated strategy, it can also be expected that there will arise a conflict of objectives because when traffic per individual peer is minimised, load is transferred to other peers, which can increase the system-wide traffic generated. When evaluating an update strategy this conflict is to be examined closely.

According to the questions asked above, there are three basic update alternatives:

- *Direct update distribution*: Here the updates are sent individually to all online trusted peers.
- *Tree-based update distribution*: Here, update information is individually encrypted and grouped together. Then, the update information is distributed over a distribution tree. This method will reduce the messages sent by peers generating the update shares. However, it is to be determined if the traffic for these peers can also be reduced.
- *Limited Update and Self-Initialisation*: In URSA it is proposed to update only a small number of peers. Using the updated peers, the shares of other peers will be recovered. This is continued and the swarming effect is exploited.

In the following, the selection of trusted peers that will start an update phase and selection of the group B that will generate the update shares are discussed. Then, the different strategies are described in more detail, and an initial analytical evaluation is performed.

The notation and assumptions in this section is consistent with the previous section (see Table 4.6). The group of peers generating update shares will be denoted as B of size β. A group of trusted peers recovering another peers P_r's share will be denoted D.

The results of the simulative evaluation will be discussed in Chapter 5.

4.6.1 Trusted Peer Selections for an Update Phase

In order to avoid chaotic update phases, there must be self-organising rules defined when and by which trusted peer an update phase should be initialised, and which group of trusted peers D should be responsible for generating the update shares. Also, when recovering a peer, it also must be determined which group of peers should be responsible for it.

When selecting an update group or a recovery group, it must be ensured that the selected trusted peers do not conspire in order to achieve knowledge that can be used to weaken the secrecy of the

scheme's private key. A group conspiring would exchange the information used to compute the update shares or the partial shares for a recovery - that is, peers would exchange the polynomials they chose for the computations. If in such a group there is only one peer that does not conspire, the knowledge of the other parts does not help an attacker to acquire some addition knowledge on the private system key. Accordingly, the update group size should be chosen according to the same considerations about the quorum size. However, here the created traffic has potentially system-wide effects. Therefore, the optimal update group size also depends on the chosen update strategy.

A way to achieve such a group is to randomly select the group peers. This concept was already applied before for the quorum selection.

4.6.1.1 Starting an Update Phase

Updates should be performed on a regular basis. In order to decide when exactly an update phase should be started, a simple rule can be applied. The rule specifies an exact period and time when an update phase should be started. An example is each Sunday, 20.00 CET, and a time server that is used as reference time.

In order to decide which trusted peer should then initialise the update phase, another simple self organising rule will specify that peer. If a Chord-Overlay (SMK$^+$01) is applied as trusted peer overlay, e.g., the active trusted peer with the highest ID could be responsible for initialising the update phase. Each peer knows its successor and predecessor and therefore can determine if it is the peer that should initialise the update. Furthermore, the neighbour peers can also determine that this peer is the responsible one. This way, even if the selected peer goes offline soon the correct replacement peer can be selected.

4.6.1.2 Selection of Update Group B

The problem of selecting the update group is similar to selecting a random quorum for signing tokens in the secure aggregation protocol (see Section 4.4). The goal of the group selection mechanism is that each trusted peer can reproduce the random number. This way, each trusted peer can check that the mechanism was executed correctly and that there is no attack on the update mechanism.

The difference from the secure aggregation protocol is that, here a swapping peer and the corresponding aggregation account is missing as a base for the computation of a random value. As a replacement, the initialising peer will query a pre-defined time server and request a millisecond-exact time stamp. As the message delay varies, the value of the time stamp will also vary. Therefore, the Hash-value of the time stamp is truly random. It can be used to determine the random update group according to the secure aggregation protocol (see 4.14).

4.6.1.3 Selection of Recovery Group D

The selection of a recovery group can be performed similar to the selection of a random quorum described in Section 4.4, with some small adaptations. The system key maintenance is performed entirely in a separate overlay network with its own ID space. In this overlay network, there are no aggregation accounts that can be used as a source to generate a truly random number.

The random quorum selection in Algorithm 4.14 uses the aggregation account of the swapping peer for generating a random number that determines the quorum. An adaptation to this mechanism, requires that first there be a random peer determined in the normal peer overlay that exists, because then an aggregation account for that peer also exists. First, the trusted peer to be recovered P_r, will lookup its trusted peer ID in the normal peer overlay. The responding peer will be selected as the aggregation

account to be queried in order to determine a random number. P_r will use this number in order to determine D.

Each trusted peer in D can repeat this process in order to determine the correctness of $D's$ recovery group selection. In order to allow a repetition of the process, the aggregation account will buffer the resulting random number for a short period in time.

If the trusted peer overlay uses a different ID space then the normal peer overlay, P_r will pad its trusted peer ID with zeros in order to allow a lookup in the normal peer overlay.

4.6.2 Direct Update Distribution

4.6.2.1 Direct Distribution Strategy

Each peer in B computes all shares for all trusted peers in the system. Then it sends individual messages to all T trusted peers over encrypted channels.

The advantage of this solution is its simplicity in the protocols.

4.6.2.2 Analytical Evaluation of Direct Distribution

Obviously, the traffic generated by this strategy is highly uneven. Also, the system-wide traffic grows linearly with the update group size.

Unverified Update Distribution

Each peer in B will have to send $n-1$ messages of with payload $(r + l_{del} + l_{double} + l_{del})$ including the update share and the update version number. For unverified updates, one IP packet can carry all of the payload. Accordingly, the overall traffic per peer in B results in:

$v_{update,nv} = (T-1) * (l_{IP} + l_{TCP} + (r + l_{del} + l_{double} + l_{del}))$

This is 50.78 kBytes per peer P_b in B for Threshold BLS updates using the settings in Table 4.6. Assuming an update group size $\beta = 5$, this results in 253.35 kBytes total traffic.

If the messages are not distributed over an encrypted channel, the updated information must be transferred encrypted. The traffic for a peer P_b within B results in (because $r_{RSA} > r_{BLS,\,q}$):

$v_{update,nv,enc} = (T-1) * (l_{IP} + l_{TCP} + (r + l_{del} + l_{double} + l_{del}))$

This is 156.25 kBytes per peer P_b in B for encrypted Threshold BLS as well as for unencrypted and encrypted URSA updates using the settings in Table 4.6. Assuming an update group size $\beta = 5$ this results in 780.47 kBytes total traffic.

Verified Update Distribution

For a verified update, using Threshold BLS the message payload results in

$l_{BLS,\,renewal} = r_{BLS,\,q} + l_{del} + (t-1)(r_{BLS,\,p} + l_{del}) + l_{del} + l_{double} + l_{del} + r_{RSA} + l_{del}$. This results in $(22 + (t-1) \cdot 66 + 142)$ Bytes for the settings given in Table 4.6. Accordingly, in Threshold BLS up to a threshold t of 21 all update information can be sent within one IP packet. This maximum threshold can be considered sufficient for the token-based accounting scheme (see Section 5.2.1). The traffic per peer in B for verified updates results in

$v_{update,v} = (T-1) * (l_{IP} + l_{TCP} + r_{BLS,\,q} + l_{del} + (t-1)(r_{BLS,\,p} + l_{del}) + l_{del} + l_{double} + l_{del} + r_{RSA} + l_{del})$

This is 1.18 MBytes per peer P_b in B for the chosen settings in Table 4.6.

The system-wide traffic is the individual peer's P_b traffic multiplied by the update group size β. Assuming an update group size $\beta = 5$ this results in 5.91 MBytes total traffic.

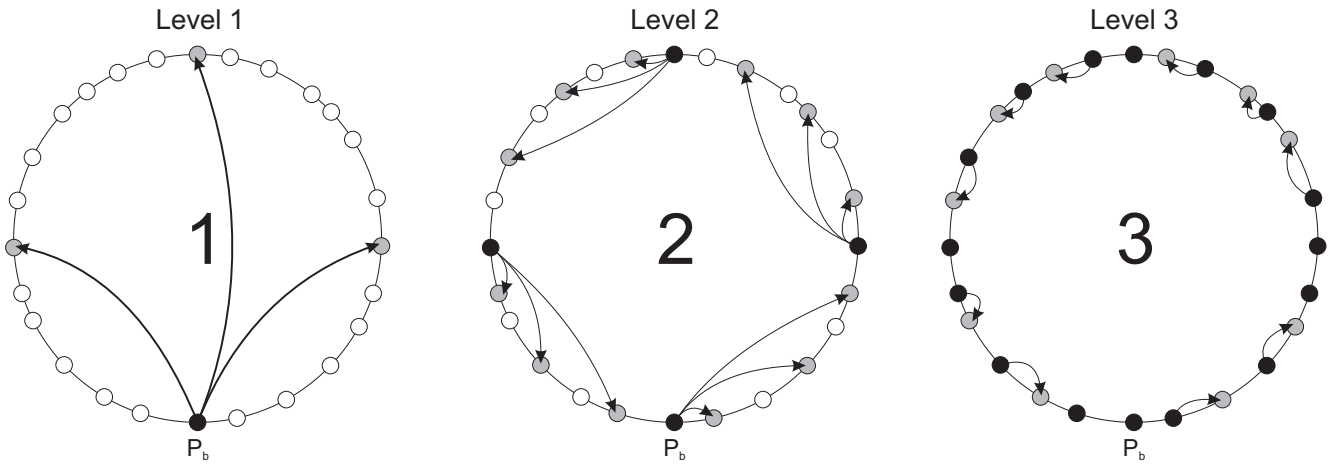


Figure 4.8: Tree-based system key update distribution strategy

4.6.3 Tree-based Distribution

4.6.3.1 *Tree-based Update Distribution Strategy*

In the tree-based distribution, each peer P_b in the update group B first computes the shares for all peers P_i. It encrypts each share with the corresponding peer's P_i public key e_i. Thus, there is no difference in traffic between URSA and BLS. Then it creates messages that contains the update information for several peers. The update information per peer consists of the peer's ID and the encrypted share.

P_b now divides the trusted peers' overlay into τ ID-ranges that should have the same size. Then it partitions the update information accordingly, and sends each of the partitions to one peer within the ID-ranges. Also, it includes the ID-range and the update version as information.

The receiving peers will extract their share, distribute the received ID-range into τ sub-ranges and distribute parts of the update information accordingly. Figure 4.8 depicts the tree-based distribution strategy.

Using this method, it can be expected that the load for the peer P_b in B can be reduced.

4.6.3.2 Analytical Evaluation of Tree-based Update Distribution Strategy

As the information in the tree-based distribution strategy will be distributed encrypted with RSA, there is no difference in the created update traffic between Threshold BLS and URSA. There is only a difference if verifiable updates are done. Here the most traffic reduction for P_bs is expected.

Unverified Tree-based Distribution

The traffic per peer P_b results for τ partitions to:

$l_{tree,nv}(\tau) = \frac{(T-1)}{\tau}(l_{ID} + l_{del} + r_{RSA} + l_{del}) + 2l_{ID} + l_{del} + l_{double} + l_{del}$

$v_{update,nv} = \tau(l_{tree,nv}(\tau) + ceil(\frac{l_{tree,nv}(\tau)}{MTU})(l_{IP} + l_{TCP}))$

where $ceil(x)$ is he smallest integer not less than x; formally: $ceil(x) = min\{n \in \mathbb{Z} | x \leq n\}$.

Verified Tree-based Distribution

Verified update traffic additionally contains $t-1$ verification values.

$l_{tree,v}(\tau) = \frac{(T-1)}{\tau}(l_{ID} + l_{del} + r_{RSA} + l_{del}+) + 2l_{ID} + l_{del} + l_{double} + l_{del} + (k-1)(r_{BLS,p} + l_{del}) + l_{del} + r_{rsa} + l_{del}$

$v_{update,v} = \tau(l_{tree,v}(\tau) + ceil(\frac{l_{tree,v}(\tau)}{MTU})(l_{IP} + l_{TCP}))$

System-Wide Traffic Optimisation

It is obvious that the traffic is lowest for P_b if $\tau = 1$, because here the verification information needs to transferred only once. However, if each peer P_i had to forward almost the complete update information to the next peer (it does not need to forward its own update share information) this solution would create the most traffic system-wide. Therefore, modelling this scenario, it can be determined the optimal τ where system-wide traffic is minimised. The analytical model yields a $\tau = 9$ for $T = 1000$ trusted peers and the settings from Table 4.6 for $t = 7,\ 10,\ 24,\ 17$.

For distributing unverified shares of one peer P_b this is 147.04 kBytes upload traffic for P_b and system-wide upload traffic of 307.52 kBytes. Using $\tau = 9$ is only a 0.34% upload traffic increase for P_b compared to the peer-minimum upload traffic with $\tau = 1$.

Assuming an update group size $\beta = 5$ the system-wide unverified update traffic adds up to 1.50 MBytes.

For distributing verified shares for one peer P_b with $t = 17$ the upload traffic results in 157.49 kBytes for P_b and system-wide upload traffic of 639.22 kBytes. Using $\tau = 9$ is only a 6.57% upload traffic increase for P_b compared to the peer-minimum upload traffic with $\tau = 1$. Furthermore, in comparison to direct verified share distribution for P_b 87.80%, traffic could be saved and system-wide 47.21% traffic could be saved.

Assuming an update group size $\beta = 5$ the system-wide verified update traffic adds up to 3.12 MBytes.

The tree-based update distribution requires many RSA encryptions. In (BKLS02) signing with a 1024 bit RSA key took 7.9 ms on a PIII 1GHz machine. As encrypting information shorter than the RSA key length is the same operation as signing in RSA, on modern desktop type computers, the tree-based update is possible to compute in a reasonable time for the token-based accounting scheme.

4.6.4 Limited Update and Self-Initialisation

The URSA update concept was presented in (LKZ+04, LL00).

4.6.4.1 Limited Update and Self-Initialisation Strategy

The URSA strategy is first to update an initial set of peers P_i. This initial group is denoted E of size ε. Note that $\forall P_b \in E$. Each peer P_b generates an update polynomial and computes the update shares for all P_i and distributes their shares to them. Each P_b now updates its share and deletes its old share as well as the received update shares and its update polynomial.

In the second step of the update phase, group I recovers the new shares of other trusted peers. As soon as a peer is recovered, it can also help to recover other peer. This way a kind of a swarming effect of share recoveries is initialised. Figure 4.9 depicts this strategy. The update traffic is distributed fairly among the trusted peers. Also, the protocol has natural robustness against offline trusted peers.

4.6.4.2 Analytical Evaluation of Limited Update and Self-Initialisation

This strategy employs direct distribution for the update group B of size β to all peer P_i in E. For example, for $\epsilon = 17$ for each P_b this traffic results in 0.81 kBytes for unverified Threshold BLS updates, 19.38 kBytes for verified Threshold BLS updates, and 2.50 kBytes for URSA updates.

Afterwards, the swarming recovery process is started. For traffic analysis it is assumed that depending on the number of peers that have already been updated each of them is chosen with an equal probability to recover the next trusted peer. For example, with $\epsilon = 17$ the P_bs will on average participate in 70 recoveries. Accordingly, using unverified Threshold BLS each P_b has to upload in total (update and self-

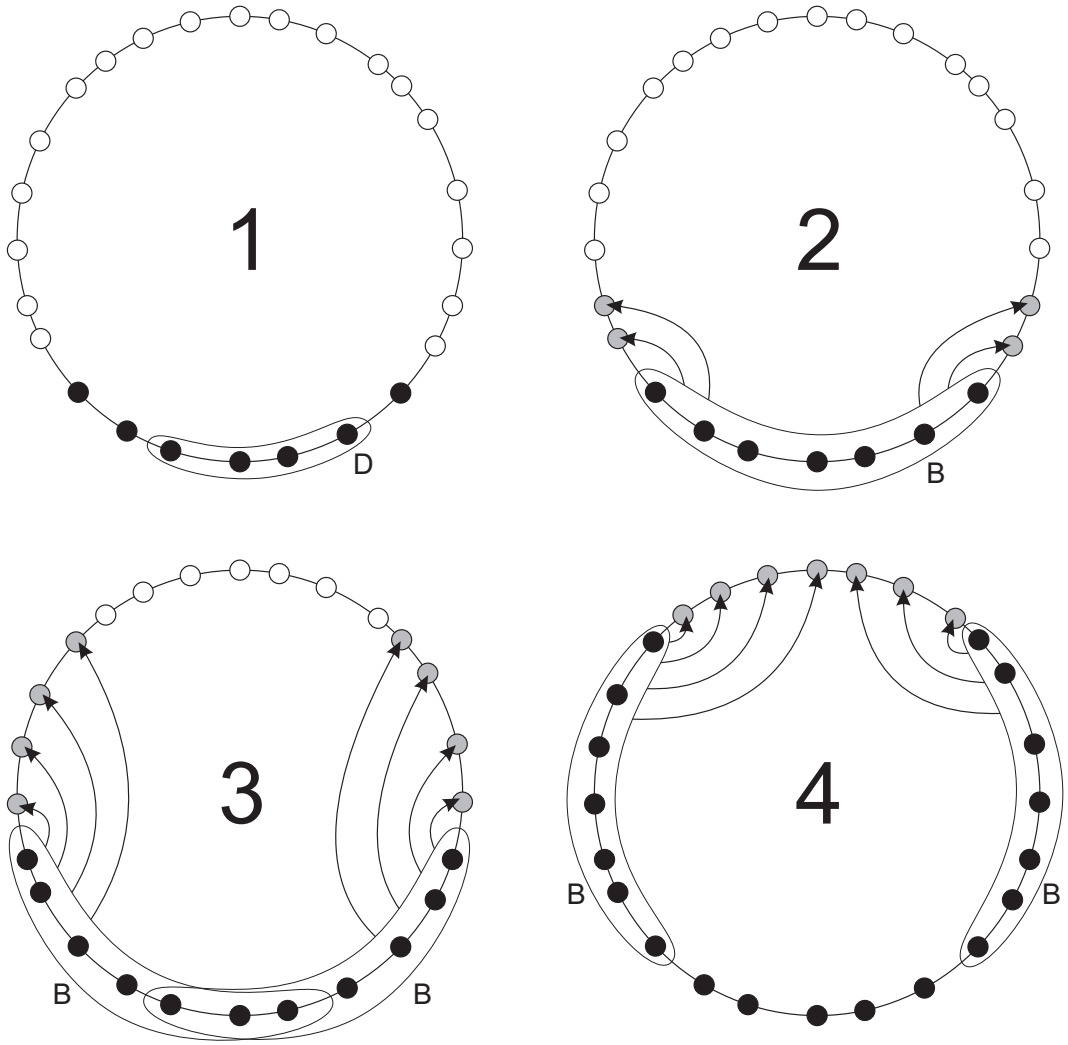


Figure 4.9: Limited update and self-initialisation distribution strategy

Table 4.7: System Key Update - Maximum Peer Traffic Comparison for $T = 1000$ and $t = 17$

	Direct	Tree-based	Limited & Self Init.	Improved Limited & Self Init.
URSA	156.03 KB	326.50 KB	188.44 KB	
BLS UV	50.73 KB	307.52 KB	61.24 KB	
BLS V	1209.73 KB	157.49 KB	1537.09 KB	380.73 KB

initialisation) 61.24 kBytes. With verified BLS the traffic results in 1537.09 kBytes. This is due to the verification values that are different for each recovery. Using URSA, the traffic per P_b results in 188.44 kBytes.

If a recovery group recovers several peers at the same time, traffic could be reduced because the verification values need to be exchanged only once within the recovery group D. Also, if the initial update group updates more than just the required quorum size the overall traffic can be reduced. Furthermore, the robustness of the recovery process will be increased. For example, if the initial update group updates $2t$ peers and also each recovery group always recovers 10 peers using the same generating polynomials, system wide traffic results in 380.87 kBytes for verified BLS. This is a reduction of 70.66%.

Furthermore, it is not fair to compare these values with the other two distribution strategies, because for them, the traffic of using only one update polynomial was presented; the size of the update group β was not considered. However, share recovery is independent of the size of the update group β.

4.6.5 Summary

In summary, the three different update strategies – direct distribution, tree-based distribution, limited update and self initialisation – the system wide traffic for an update group sizes of β =5, 10, 15 will be compared. The trusted peer system has a size of $T = 1000$; the threshold is set to $t = 17$. The initial group size ϵ for the limited update and self initialisation strategy is set to $\varepsilon = 50$.

Table 4.7 shows the resulting traffic for a peer P_b in an update group B. However, as mentioned above, this comparison is not fair for the “limited update and self initialisation” strategy, because all peers (apart from the initial updated distribution group E) have received their complete new share, whereas for the other strategies, only the update of one peer in B was considered. Also, it has to be taken into account that this strategy distributes the generated traffic over a larger period of time than the other two strategies. Furthermore, the improved limited self initialisation strategy creates significantly less traffic. The tree-based distribution especially creates all traffic immediately.

The system-wide traffic is compared in Table 4.8. Here, the “limited update and self initialisation” strategy creates the most traffic; the improved version creates significantly less traffic. Also, this strategy is only marginally sensitive to the update group size β. Therefore, for large β it can be the best solution. The tree-based distribution is the best alternative for verified updates. Direct distribution should only be chose for small update sizes and unverified traffic.

This analysis shows that the “limited update and self initialisation” strategy seems to create the most traffic by far. However, the other update strategies require the knowledge of all system trusted peers; tree-based distribution also requires the knowledge of all trusted peers’ public keys. This assumption seems to be quite hard to actualise. In order to collect this information, additional traffic in the overlay is required. Such information could also be collected over time when trusted peers interact with each other. However, each peer in the update group meets different trusted peers over time; and a consistent knowledge is required with the update group. Especially for the tree-based distribution, the required

Table 4.8: System Key Update - Complete Traffic Comparison for $T = 1000$ and $t = 17$

	β	Direct	Tree-based	Limited & Self Init.	Improved Limited & Self Init.
URSA	5	0.76 MB	1.59 MB	2.56 MB	
BLS UV	5	0.25 MB	1.50 MB	0.83 MB	
BLS V	5	5.91 MB	3.12 MB	20.91 MB	5.74 MB
URSA	10	1.52 MB	3.19 MB	2.57 MB	
BLS UV	10	0.50 MB	3.00 MB	0.84 MB	
BLS V	10	11.82 MB	6.24 MB	21.00 MB	5.93 MB
URSA	15	2.29 MB	4.78 MB	2.59 MB	
BLS UV	15	0.74 MB	4.5 MB	0.84 MB	
BLS V	15	17.72 MB	9.36 MB	21.10 MB	6.13 MB

Legend: V = Verified, UV = Unverified

knowledge of the trusted peers' public key does not enable information collection over time. Public keys must be verified before usage with the owner, in order to guarantee an attack free update distribution.

The improved version of the "limited update and self-initialisation" strategy is a compromise here. It does not require the encryption of information, which is why it is the preferred strategy over tree-based distribution.

In conclusion, the "limited update and self initialisation" strategy seems the most realistic one to apply for the token-based accounting scheme.

4.7 Failure Recovery in Transactions

During transactions there are several non-standard situations that can happen. In such situations it can appear that one peer tries to defraud the other peer, although both peers acted in good faith. In this section, it is discussed how these situations should be handled.

4.7.1 Service Delivery Interruption

A service delivery can be interrupted for several reasons where no transaction party acts in bad faith. Such reasons are crashed computers, broken Internet connection, DSL-logout after the maximum session time and re-login assigning a different IP-address, electricity failure, etc. All of these things could happen at either transaction partner. Such failures can be of very short duration or of longer duration. The result is that there is no communication possible between the transaction partners for the duration.

4.7.1.1 Failures with Duration

In order to cope with short duration failures, there are timeouts implemented with the transaction protocol. If the failure is resolved within the timeout, the service delivery can be continued or restarted. In case of a restart, the service receiver will re-use the tokens it used before in this transaction, but it will update the transaction related information contained in the token. The service provider has to delete the tokens it received before. If the receiver does not trust the provider to do so, it can demand that the provider to add a note to the old tokens stating that they will be resent by the receiver due to a failure and sign the tokens with its private key. Then it will send these tokens back to the receiver. Accordingly, if

the receiver will be accused of double spending later, it can present the signed statement of the provider that will annul the accusation.

4.7.1.2 Failures with Long Duration

Failures with longer durations have the effect that a agreement between the transaction partners about a continuation of the service cannot be reached. If it is a commercial service that is provided, it should be agreed before within the Service Level Agreement (SLA) how to deal with such a situation. Depending on the service – commercial, non commercial, main characteristic of the service (file transmission, higher valued service like x-ray diagnosis) and on the agreed payment methodology (pre-, post payment, trustworthy transaction with partial payments), different alternatives can be chosen. A service provider could be responsible for delivering the service again, or the service receiver could loose the tokens it already paid. In general in such situations, it is important that potential double spending situations are resolved. Therefore, tokens should not be re-used. Instead, a service provider should re-fund any overcharged tokens to the service receiver by sending the receiver own tokens with the appropriate accounting data. Then, the service receiver can swap these tokens with the ratio 1:1 in the next token aggregation.

In general, if any peer feels that the rules have been breached, it should report that to the reputation system.

4.7.2 Service Cancelled

Each transaction party can cancel the transaction while it is ongoing. Typically, applications implement a method to cancel the service delivery from both sides. Therefore, a service cancellation can be clearly distinguished from a failure. Typically, the SLA should contain terms how paid tokens are to be handled. The service receiver could loose these tokens or a re-fund of part, or all of the tokens (see above).

4.7.3 Disagreement About the Transaction Status

There can exist situations where the transaction partners disagree about the status of a transaction:

- The service receiver claims that the service was not completely delivered although it was fully paid.
- The service provider claims that the service was not fully paid although it was fully delivered.

Generally, the usage of the trustworthy transaction protocol is strongly suggested, as it can avoid many such situations.

Both situations may not be able to be resolved. If such a situation exists, the transaction partners have to file a complaint with the reputation system. Peers that are involved more often in such situations will eventually have a degraded reputation value. The conclusion is that these peers create problems in the system, and they can be punished by the system in some way.

Resolving Situations Where Service Was Not Delivered Completely

In situations where the service receiver claims a service was not delivered completely, either the service receiver received the service and wants to defraud the provider, or the other way around. Using the trustworthy payment process, the complete service delivery can be comprehended except for the last part of the delivery. Here, the service provider can send this part again. Typically, this should only cost bandwidth resources, as a result of a higher level service, the service receiver should still cache. This is not too expensive. If this is not successful, probably both transaction parties will file a complaint with the reputation system.

Resolving Situations Where Service Was Not Fully Paid

Payment can be comprehended much easier than service delivery due to the information stored in the aggregation accounts. In a trustworthy transaction, the tokens to be used in a transaction are pre-registered. If a service receiver does not pay some of these tokens, it has to re-send them. The service provider can also notify the aggregation account about a received token only once. Double spending within one transaction is not possible. A malicious service provider could not notify the aggregation account about the received tokens and claim it never received them. However, a token it does not identify to the aggregation account can also not be swapped for new tokens. Therefore, the aggregation accounts should un-block tokens that have not been brought to the notice of the service provider after a specific period of time.

Accordingly, using the trustworthy transaction protocol it can always clearly be decided if the service was paid for or not.

4.8 Summary

This chapter presented the details of the protocols of the token-based accounting scheme. It covers seven different parts of the token-based accounting scheme: Peer identification, the account holder set, trusted peers, the security extensions to the aggregation protocol, system key maintenance protocols, system key update distribution, and failure recovery in transactions.

In the first section, it was discussed how peers get assigned a permanent overlay ID. This is resolved by using a hash value of a peer's public key.

The second section is concerned with mechanisms required for a viable operation of the account holder set. It is divided into four parts. In the first part it is discussed where account holder sets are located, how peers are assigned to an account holder set, and how peers access and post information to aggregation accounts. These mechanisms form a novel anonymous access mechanisms to aggregation accounts where the account holders of aggregation accounts stay anonymous, even as the accessing peer can be clearly identified. If the receiver is hidden from a message sender, trials of cheating and collusion attacks as well as malicious behaviour are impeded, and as such can be traced back to its source. Existing solutions in the area of anonymous communication could not be applied, because these require that both sender and receiver are anonymous. The developed protocol for receiver anonymity with sender identification is applicable to all structured p2p overlay networks that form a ring topology. The second part presents the protocols used to maintain account holder sets under churn. Here, the protocols are described that detect the actual size and location of an account holder set, that lock an account holder set so maintenance actions can be performed without creating inconsistencies, that move an account to a new position, and how peers perform a graceful handover of their accounts before they log off. The third part describes the information stored in an account holder set and how it is accessed. The fourth part discusses the alternatives for consistency mechanisms used for aggregation accounts. A majority-based consistency mechanism is selected.

The third section is concerned with trusted peers. It discusses how trusted peers are organised, assigned and excluded from the trusted peer overlay network. Trusted peers are organised in a separate overlay network in order to allow efficient key maintenance operations. Also, a mechanism is presented that is used to find trusted peers in the general system overlay. It is also discussed how incentives can be given to peers to motivate them to act as trusted peers.

The fourth section presents an extension to the aggregation protocol, that enhances its trustworthiness. The trustworthy aggregation protocol includes a novel mechanism that selects the aggregation administering peer as well as the quorum randomly, and permits to verify the randomness of the selection afterwards by any peer, although the result of the selection process cannot be determined in advance. This allows the system to review whether the protocol review was truthfully executed. This prevents cheating and collusion during token aggregation.

The fifth section discusses the usage of proactive secret sharing techniques in order to ensure the secrecy of the scheme's private key over a longer period of time. Proactive secret schemes are discussed for the two selected threshold cryptography schemes Threshold BLS and URSA. This section presented the distributed key generation and share assignment phase, the share recovery phase that is used to create new shares for new trusted peers as well as to recover trusted peers that possess an outdated share, and the update phase where update shares of the system key are distributed. For the different phases an analytical traffic assessment is performed for both threshold schemes. In general, URSA currently does not provide the possibility of using verified sharing techniques. Therefore, BLS is the preferred threshold scheme that should be applied.

The sixth section analyses different key distribution strategies in terms of generated traffic. The strategies are used to update the complete trusted peer overlay with update shares. The best strategy was an improved version of the limited update and self initialisation method used for URSA was identified.

The seventh section discusses failure recovery in transactions. Here situations of recovery during a transaction where the service delivery is interrupted or cancelled, as well as situations where there are disagreements about the transaction status are discussed.

In general, this chapter focused the trustworthiness of the token-based accounting scheme. If all mechanisms presented are applied in the token-based accounting scheme, there remain two situations where fraud is possible by injecting forged tokens. Either a peer receives knowledge of the scheme's private key and can control the majority of its account holder set, or a peer achieves the dishonest cooperation of t trusted peers[5] and the majority of its account holder set. However, in both situations, the required and verifiable randomness of the aggregation administrator and quorum is violated. Therefore, this fraud is recognisable, even though this requires some effort.

[5] t denotes the threshold of the applied threshold cryptography scheme.

5 System Evaluation

When designing a mechanism for peer-to-peer systems, it must be shown that this mechanism will work as conceptualised and further what the mechanism's performance will be. The communication such mechanisms employ is asynchronous. Therefore, it is hard to observe and control such a system. This makes the performance analysis of a real life system difficult. Hence, it is important for mechanisms designed for decentralised autonomous systems to be evaluated before roll-out, using a suitable evaluation technique.

This chapter presents the in-depth evaluation performed on the token-based accounting scheme using analytical techniques as well as simulations. Using analytical techniques the performance of the token-based accounting scheme is assessed depending on the configuration parameters. Furthermore, the traffic created by the developed protocols is estimated. Using detailed simulations the traffic generated by the token-based accounting scheme is investigated and the influence of the configuration parameters is analysed. The analytical results serve as control for the simulation results.

This chapter is structured into eight sections. First, the evaluation goal is stated and the evaluation criteria and metrics are developed. From this the required evaluation methods are deduced in Section 2. Section 3 presents analytical results about the message overhead of the developed protocols. Section 4 describes the simulation model and the simulation scenario employed to evaluate the token based accounting scheme. Then the in-depth analysis of the simulation results is presented. Section 5 describes the simulation model and simulation scenario used to evaluate the overhead introduced by the system key management protocols. Section 6 presents an analytical comparison with KARMA, which is the related work most similar in its performance to the token-based accounting scheme. Section 8 concludes this chapter.

5.1 Evaluation Criteria and Methodology

When evaluating a mechanism in the context of multimedia communications the efficiency of the mechanism must be assessed. The efficiency is expressed as the ratio of performance to costs (SHL$^+$06). Looking only at a mechanisms performance might lead to a system with excellent performance but prohibitive high costs for real world deployment. Accordingly, for the token-based accounting scheme two main criteria exist in order to assess its efficiency.

For an accounting scheme for decentralised autonomous systems the trustworthiness of the information administered by the scheme presents the systems performance. Parameters and metrics about trustworthiness are discussed in Section 5.1.1.

The costs introduced by the accounting scheme are the communication costs created. Accordingly, for a given level of performance, the introduced costs must be determined in order to evaluate the token-based accounting scheme. Furthermore, within peer-to-peer research the most important evaluation criterion for a mechanism is its scalability. Scalability describes the quantitative adaptability of a system or mechanism, i.e. the adaptivity to a changing number of entities/nodes or offered services in the system (SHL$^+$06). Thus, a main parameter for the scheme's evaluation is the system size and how the costs develop with changing system size. Obviously, both communication costs and scalability, are closely related. Their parameters and metrics are discussed in Section 5.1.2.

5.1.1 System's Performance

The token-based accounting scheme's mechanisms for achieving trustworthiness have been discussed in depth in the preceding chapters. Here, the main concepts are summarised in order to conclude the crucial parameters and metrics for evaluating the system's trustworthiness.

The trustworthiness of information administered by the token-based accounting scheme depends mainly on three elements:

- the usage of threshold cryptography for creating system's signatures under issued tokens using the Token Aggregation Protocol,
- periodical updates of the key shares distributed,
- the account holder set, that holds information about which tokens have been issued, have been spent, are intended to be spent.

5.1.1.1 System's Performance Parameters

Token Aggregation Protocol

Issued tokens must be signed with a system wide key by a quorum of trusted peers using threshold cryptography. Using threshold cryptography, only unanimous quorum decisions lead to valid token signatures. Therefore, the probability of false positive signatures, i.e. falsified signatures, decreases with increased quorum size. Thus, the first parameter for influencing the token-based accounting scheme's trust is the quorum size t. How the optimal quorum size t can be determined will be discussed in Section 5.2.1.

As the token aggregation protocol is the scheme's main mechanism in order to avoid forged tokens, avoiding false positive signatures has absolute priority over avoiding false negative signatures, i.e. tokens that should have been issued were not issued due to a false decision of a quorum member. If a peer does not receive tokens through a token aggregation process due to a false negative decision, the peer can always repeat the aggregation process, when the second time, the quorum will consists of different peers, due to the quorum selection process, described in Section 4.4.

Periodic Key Shares Updates

A key share update invalidates old key shares. With a mixture of old shares and new shares tokens cannot be signed. How often a key share update should be carried out depends on the minimum time an attacker would require to receive knowledge of $t-1$ key shares. Determining this time is a very complex task and it requires many assumptions that are application dependent. Thus, determining the optimal frequency of key share updates is beyond the scope of this dissertation.

Account Holder Set

The account holder set has the task of checking for behaviour against the rules. In particular, double spending and introducing tokens that have been signed not using the Token Aggregation Protocol. Account information is replicated over a set of peers. Therefore, another performance parameter of the scheme is the Account Holder Set Size k.

As described in Section 4.2, transactions are possible even when account information might be lost. Thus, information held in an account holder set must be available for the vast majority of time. In conclusion, another performance parameter in the context of the account holder set size is the probability p_{ahs} an account is available.

The consistency mechanism for the account holder sets depends on simple majority. In order to have a clear decision about account information an account holder set requires at least two peers with consistent

account information. We define account availability as at least two peers must holding consistent account information. Accordingly, the account holder set size k must be chosen in a way that accounts are available at least with probability p_{ahs}.

Furthermore, in order to evaluate the account holder sets performance, it must be determined with which probability p_{cor} information held at a set is correct. Incorrect information can occur due to lost information due to churn that the mechanisms built around the account holder sets cannot resolve. An example is the execution of the consistency mechanism that leads to false account information.

5.1.2 System's Cost

The costs that token-based accounting scheme bears are the communication costs between the participating nodes, the computation costs, and the storage costs at the participating nodes. Measurements with the token-based accounting prototype showed that 50 foreign tokens require a storage space of approximately 53 kBytes. Creating and verifying RSA signatures creates negligible computational costs. So if we assume that the token-based accounting scheme will be coupled with peer-to-peer application that typically run on desktop class PC, computation and storage cost can be disregarded. In conclusion, in this dissertation only the communication costs in terms of created traffic will be analysed.

When analysing the traffic created, two different perspectives are important. The first is the relation of the overall peer-to-peer traffic created by the application to the traffic generated by accounting services. The second is the impact of accounting traffic on an individual peer. It must be analysed whether peers can become overloaded by the accounting traffic. Therefore, the traffic generated at single peers must be analysed.

5.1.2.1 System's Cost Parameters

The parameters influencing the performance of the token-based accounting scheme also will influence the traffic created. The larger the quorum, the more traffic is created during an token aggregation process. The larger the account holder set, the more traffic is generated at transactions and token aggregation. The number of trusted peers T determines the traffic created during for key share updates and also influences the distribution of traffic among the peers. The more trusted peers there are, the less likely a specific trusted peer is selected for aggregation, etc. Therefore, it can be expected that the traffic will be distributed more fairly among the peers with a larger number of trusted peers. Accordingly, an important system parameter is the ratio p_{tp} of normal peers to trusted peer in the system.

There are further system parameters that influence the amount of traffic created. The overlay network used for lookups creates traffic for lookups and for maintenance. Furthermore, peer behaviour also has influence on the amount of maintenance traffic created by the overlay network and by the account holder set. churn is especially important here. Churn is influenced by whether the peers conduct a graceful handover of information to other peers before they leave, or if they just vanish from the system. The size of the peer-to-peer system also influences the traffic stemming from these sources.

Other user behaviour also influences the amount of traffic generated, however this is largely independent of the system size. This is the frequency users perform transactions and token aggregation.

5.1.3 Evaluation Techniques

According to Jain (Jai91), the alternative evaluation techniques are analytical modelling, simulation, and measurement. The discussion of the parameters show that potentially different evaluation techniques should be applied.

Performance Parameters

The exact value of the quorum size can be calculated using an analytical model. This will be presented in Section 5.2.1.

The account holder set size could also be determined using an analytical model. Here, queueing models could be applied. However, these models are well understood for problems that can be modelled by the exponential probability function for inter-arrival and service process. However, in peer-to-peer systems, several measurement studies have shown that the inter-arrival and also the service process are not exponentially distributed (see e.g (BQ04), (SR05), (SENB07a)). Accordingly, for this problem simulation is the appropriate evaluation technique (see Section 5.2.2).

Cost Parameters

The traffic generated by the token-based accounting scheme varies in the way it can be analysed.

Traffic generated from transactions and token aggregation is largely independent from complex influences like churn or system size. Thus, it can be determined quite exactly using an analytical model. As a prerequisite, the token-based accounting scheme's performance parameters must be known. They will be determined in Section 5.2.

The remaining elements of the token-based accounting scheme, i.e. all mechanisms in the context of the account holder set, have closer interdependencies with the overlay network and churn. Here, due to the difficulty in modelling the account holder set under churn (see above), an analytical model cannot be applied.

Furthermore, if the whole system should be evaluated, a technique should be chosen that is able to capture all elements of the accounting scheme.

One goal of the evaluation is also to determine the scheme's scalability. Therefore, we prefer simulation over measurements using a prototype as evaluation technique.

5.1.4 Selected Metrics

Performance Metrics

As the quorum size determines the trustworthiness of the aggregation protocol, there are no performance parameters that will be evaluated in context of the token aggregation protocol.

In context of the quorum size, there are several parameters that determine the performance of this mechanism. The account holder set varies over time due to churn, but consistency of the information held is required. As described in Section 5.1.1.1 an account holder set size of at least 3 is required with two of the accounts being consistent. In the evaluation it is to be determined at what percentage this condition is violated. This should be measured with different parameters influencing it, as described in Section 5.1.1.1.

Cost Metrics

The basic cost metric is the traffic created. It is to be determined system-wide in total and split up for the different mechanisms of the token-based accounting scheme. Furthermore, the traffic per peer should be determined in order to understand how fair the traffic is distributed. In addition, peers could be overloaded. Therefore, their download queue and upload queue is relevant.

Finally, a comparison between the accounting traffic and the traffic the services accounted for create, is meaningful.

5.2 Trustworthiness Analysis

In this section the values for the performance parameters of the token-based accounting scheme are derived. These are the quorum size and the account holder set size.

5.2.1 Quorum Size

The quorum of size t is randomly chosen from all trusted peers T. In order to avoid false positive signatures, at least one trusted peer must be chosen who behaves according to the rules. We call such peers "good peers" in the following. The percentage of good peers among all trusted peers is denoted p_g. p_g is a system variable that needs to be estimated.

The hyper-geometrical distribution describes the problem of randomly choosing a quorum out of all trusted peers, where only a specific percentage of trusted peers is good. The resulting Equation 5.1 describes the achieved Trust Level L for given total number of trusted peers T, a specific percentage of good peers among them p_g, and a given quorum size.

$$L(T,t,p_g) = \frac{\binom{T \cdot (1-p_g)}{t}}{\binom{T}{t}} \tag{5.1}$$

where

T	number of trusted peers
t	quorum size
p_g	percentage of good peers

Figure 5.1 a - d summarise the resulting quorum sizes for example values. These results will be used to determine the quorum size for the complete system evaluation.

5.2.2 Account Holder Set Size

The determination of the optimal account holder set size is crucial for the availability of aggregation accounts as well as for the reduction of costs stemming from the maintenance of account holder sets. As discussed in Section 4.2.1, the protocol research by Knežević in (Kne07) could be applied as an adaptive mechanism. However, this mechanism does not currently consider account maintenance mechanisms like those applied in the token-based accounting scheme. Therefore, these results cannot be applied to determine an account holder set size to be used in the evaluation of the token-based accounting scheme.

Also queueing theory is not yet applicable to model the account holder set maintenance mechanisms using actual churn models resulting from measurements (see (BQ04, Dar05, SENB07b, SENB07a)); these suggest modelling churn using either a mixed log-normal distribution ((Dar05)) or a Weibull distribution (SENB07a, SENB07b)).

Accordingly, in order to determine the optimal account holder set size the mechanisms for maintaining its size were simulated in the presence of churn. An abstract simulation model was created simulating the core maintenance mechanisms for checking the account holder set size, moving accounts, and handover of accounts.

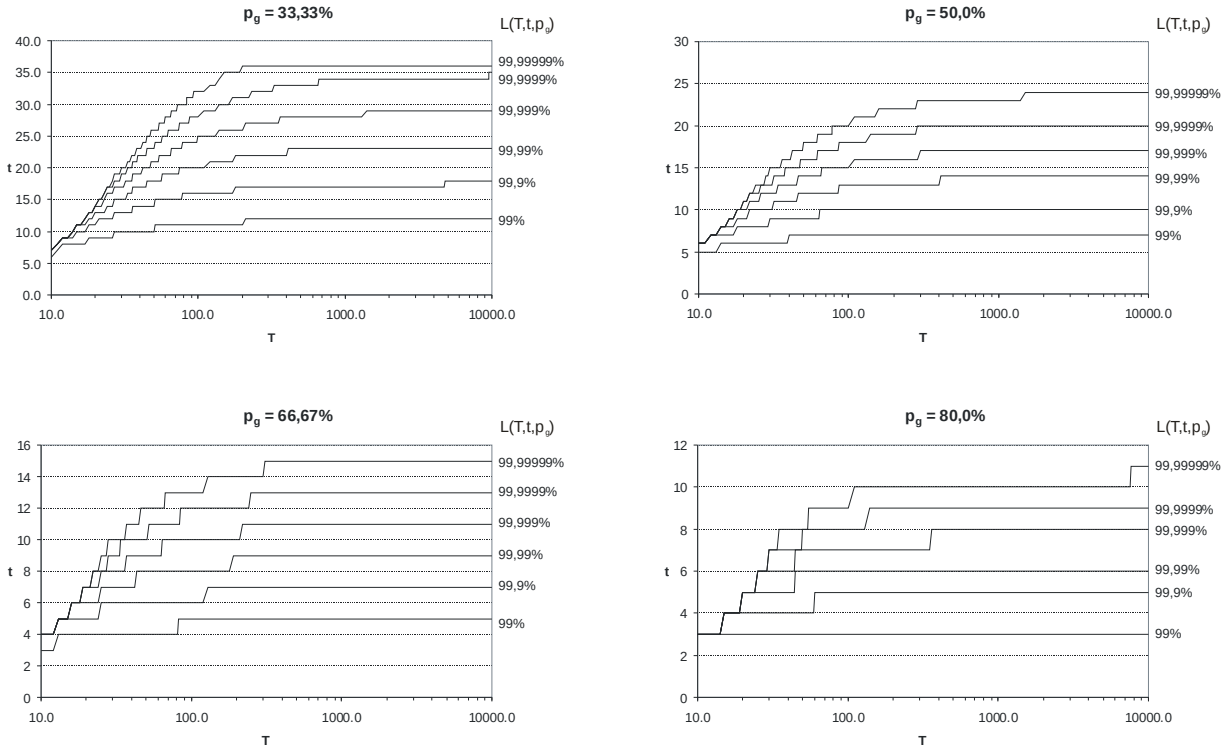


Figure 5.1: Required quorum size according to equation 5.1

5.2.2.1 Simulation Model

The simulation models the following mechanisms under churn:

- Graceful handover: when a peer leaves, a new account holder is selected for all accounts it is responsible for.
- Peer leaves without graceful handover: the peer will not be replaced.
- Detection of replica number: If an account has less then the default number of account holders, the missing number of holders is added.
- Detection of correct account holder set position: The account holder set is relocated and the correct number of account successive holders is assigned.

The graceful handover happens when a peer leaves with probability p_{gh}. The replica number of any account is checked with a frequency t_{cz}. The correct account holder set position is checked with a frequency t_{cp}.

In the simulation t_{cz} was set to 5 minutes, and t_{cp} was set to 10 minutes. A peer performed a graceful handover with p_{gh}= 90% or p_{gh}= 50%.

For churn two different models were applied. Due to measurements from (SENB07b) both peer lifetime and peer offline time are Weibull distributed. The corresponding distribution fit is given in Table 5.1. The second churn model stems from the measurements from (BQ04). Here, the fit from (Dar05) consisting of a mixture of two log-normal distributions is applied for lifetime and offline time. The corresponding distribution fit is given in Table 5.2. In order to analyse a broader set of churn scenarios, both churn models are modified with the churn factors $d_{var} = \{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}\}$.

In order to fully observe the effects of churn, each experiment simulated one week of real time.

Table 5.1: Churn According to (SENB07b): Weibull Distribution Parameters

	scale λ	shape k
Lifetime	0,61511	169,5385
Offline time	0,47648	413,6765

Table 5.2: Churn According to (BQ04, Dar05): Mixed Log-normal Distribution Parameters

Life time & Offline time		
Weight	μ	σ
0.8	8.39	0.97
0.2	9.57	1.67

As scalability does not have an effect on the results for this simple simulation model, only one system size of 1000 was simulated.

Each experiment was repeated 20 times.

5.2.2.2 Simulation Results

The account holder set is an additional security mechanism. It is sufficient if users have to assume that account holder set information is available. Thus, a very high availability must be reached by the account holding set mechanisms, but not complete availability. Therefore, the goal was a probability that at least 3 account holders exist for each account p_{ahs} = 99%.

Table 5.3 summarises the simulation results, where k denotes the configured account holder set size and $a_{\tilde{k_n}}$ denotes the accumulated time a specific account holder set size n was observed in the simulations. Furthermore details are listed in the Appendix in Section E.1.

Table 5.3 lists the accumulated time an account holder set size of 3 and 4 were observed on average. This is summed over all account holder sets. Accordingly, a value of 139.25 seconds can mean that either one account holder set had a size of 3 for this time or there were several account holder sets that had a size of three for a shorter time, which in sum results in 139.25 seconds. Figure 5.2 shows for example, the average number of account holder sets with size 3, 4, and 5 for the experiment Weibull, $d_{var} = 0.5$, $p_{gh} = 0.9$.

In the experiment for Weibull, $p_{gh} = 0.9$, $d_{var} = 1$ an account holder set size of 3 did not happen; however a set size $\tilde{k_3}$ did happen so often, that an account holder set size of 6 is the more reliable result.

5.2.2.3 Summary

In order to determine the required account holder set size, an abstract simulation model was built. It simulates churn, as well as the account holder set size maintenance mechanisms that repair account holder sets. Eight different churn models have been simulated with two different probabilities for account handover when a peer leaves.

The result is that depending on the churn model, the account holder set size should be configured to values from $k = 6$ to $k = 10$.

5.2.3 Summary

In this section the two main configuration parameters of the token-based accounting scheme were observed that influence its performance and costs.

Table 5.3: Required Holder Set Sizes For Different Churn Models

Distribution	p_{gh}	d_{var}	k	$a_{\tilde{k_3}}$ [sec]	$a_{\tilde{k_4}}$ [sec]
Weibull	0.9	1	6	0	14511.2
		$\frac{1}{2}$	6	139.25	60536.15
		$\frac{1}{3}$	6	839.05	125505.1
		$\frac{1}{4}$	7	4.2	2751.8
	0.5	1	7	91.3	6551.35
		$\frac{1}{2}$	8	20.9	2001.8
		$\frac{1}{3}$	9	0	479.3
		$\frac{1}{4}$	10	0	137.4
Mixed Log-normal	0.9	1	6	21.1	16484.4
		$\frac{1}{2}$	6	301.1	66831
		$\frac{1}{3}$	7	47.2	2089.1
		$\frac{1}{4}$	7	4.35	4554.2
	0.5	1	7	52.5	7318.5
		$\frac{1}{2}$	9	0	92.5
		$\frac{1}{3}$	9	0	545.75
		$\frac{1}{4}$	9	44.3	2220.7

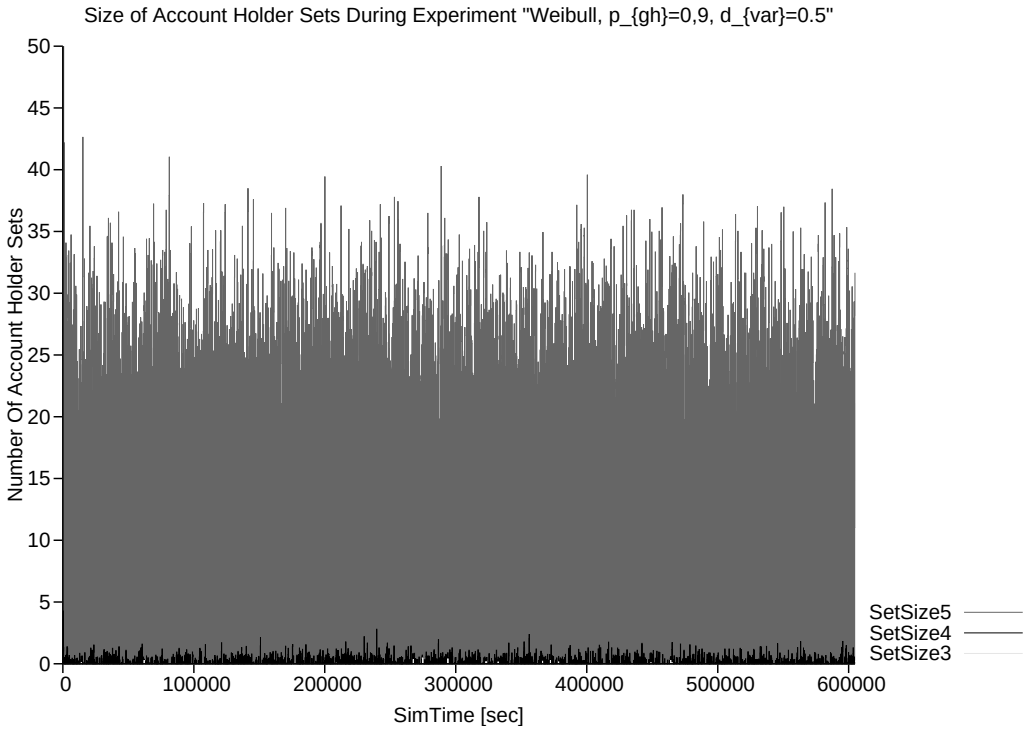


Figure 5.2: Account holder set size simulation results for $d_{var} = 0.5, p_{gh} = 0.9$

The quorum size influences the trustworthiness of the token-based accounting scheme. The trustworthiness that can be achieved with a specific quorum size depends further on the number of trusted peers present in the system, and the probability that a trusted peer acts honestly. For systems with 500 to 10,000 trusted peers, quorum sizes from $t = 7$ to $t = 17$ are required to achieve a trustworthiness of 99% to 99.999%, if at least 50% of the trusted peers act honestly. Trustworthiness of the quorum is defined as the probability that a randomly selected quorum consists only of peers that are willing to defraud.

The account holder sets host the aggregation accounts. The aggregation accounts are a mechanism that further enhances the token-based accounting scheme's trustworthiness. Therefore, this information must not be lost. In order to determine the optimal number of replications of an aggregation account the relevant maintenance mechanisms for account holder sets have been simulated in the presence of churn. The goal of the simulation was to determine the required account holder set size, so that at least three peers host a specific aggregation account. This way, the consistency mechanism that is built on majority decisions can still find the consistent aggregation account state if there is one account storing deviating account information. As result of eight different churn models combined with two different probabilities for peers handing over their account when leaving, it was found that the account holder set size k should be set to values between 6 and 10.

5.3 Analytical Traffic Analysis

Most important for the evaluation of the token-based accounting scheme is the traffic the scheme introduces into the system. Before the traffic will be simulated, a worst case assessment will be performed. There are three parts in the token-based accounting scheme that can be assessed separately. The first part is traffic created by transactions between peers, i.e., when a peer requests and receives a service from another peer, and the token aggregation processes conducted due to these transactions. The second part is the traffic created from managing account holder sets. The third part is the key management traffic created when the key shares owned by the trusted peers are updated.

For the analytical assessment, the number of messages required will be calculated. Message sizes will be regarded in the scheme's simulation.

5.3.1 Analytic Transaction Traffic Assessment

In the token-based accounting scheme, traffic created due to transactions and token aggregation processes is relatively deterministic since the majority of messages created is independent of current system size and of churn. However, the number of hops a lookup message travels in the overlay depends on system size and churn rate.

Considering the token-based accounting messages (and not considering the overlay messages) the important parameters are:

- Number of Transactions n
- Quorum Size t
- Account Holder Set Size k
- Average Transaction Value s: The average number of tokens spent by a peer in one transaction.
- Batch Size b: The average number of tokens swapped in a token aggregation process

Transaction

Messages required to request a service will not be considered in the analytical assessment, as these messages belong to the application and not directly to the token-based accounting scheme.

The traffic for one transaction using the trustworthy transaction protocol is composed of the following number of messages:

- Requester sends unsigned tokens to provider: 1
- Provider checks these tokens with the account holder set: $1 + 2k$

Now the provider starts the service provisioning.

- During the service provisioning, the requester sends token by token to the provider: s
- Optionally, the reception of a token could be approved by the provider: s

Accordingly, for one transaction the number of messages created results in: $M_{Trans}(k) = 2 + 2k + 2s$.

Token Aggregation

For assessing the traffic of one token aggregation process, it is assumed that the peer swapping tokens provided each service to a different peer. Accordingly, there must be $\frac{b}{s}$ aggregation accounts checked for double spending. Like for transactions, the overlay lookup operations are not considered in this analysis. A token aggregation process requires the following number of messages:

- Swapping peer sends b foreign tokens to random trusted peer: 1
- The trusted peer queries the account holder set in order to determine the administrating trusted peer: $1 + 2k$
- The trusted peer forwards the foreign tokens to the aggregation administrator: 1
- In order to detect double spending, the aggregation administrator queries all aggregation accounts of the owners of the foreign tokens: $\frac{b}{s}(1 + 2k)$
- The administrator forwards the fresh tokens to the quorum: t
- The quorum peers update the swapping peer's aggregation account: $t(1 + 2k)$
- The quorum sends the partially signed tokens to the swapping peer: t

For one aggregation process, the number of messages created results in: $M_{Aggr}(k, t, b) = 2 + (1 + t + \frac{b}{s})(1 + 2k) + 2t$

Complete Transaction Dependent Traffic

The traffic created due to transactions and aggregation processes is not dependent on time but solely on the number of transactions performed by the system's peers. An aggregation process will be performed on average every $\frac{ns}{b}$ transactions. Accordingly, the number of messages can be assessed using Equation 5.2, where n is the number of transactions performed.

$$M(n, k, t, b) = n(2 + 2k + 2s) + \frac{ns}{b}(3 + \frac{b}{s} + 3t + 2tk + 2k(\frac{b}{s} + 1)) \tag{5.2}$$

5.3.2 Analytic Account Holder Set Management Traffic Assessment

Unlike the transaction related traffic, the account holder set management traffic depends on churn of peers and on time, because accounts are handed over when peers leave and there are frequent actions controlling and repairing account holder sets.

As described above, churn in p2p systems can not be described using exponential functions. Therefore, Markov Chains and queueing theory cannot be applied to model the traffic created from churn. This part of traffic will only be regarded in the simulation.

Account holder set management consists of two mechanisms.

- Detection of replica number and adding account holders if necessary.
- Detection of correct account holder set position and moving it, if necessary.

Apart from the account holder set size (k) there are two frequencies to be determined as system parameters.

- f_{ck} is the frequency an aggregation account's set size is checked.
- f_{cp} is the frequency an aggregation account's position is checked.

5.3.2.1 Simple Mechanism Analysis

For managing the account holding sets the following mechanisms are used: Detection of replica number, detection of correct account position, account holder set locking, account consistency, account movement, and graceful account handover. The message created by the mechanisms is summarised now. The number of messages stated here is only correct if the overlay's structure is completely correct, and no involved peers fail or go offline during runtime of such a mechanism. Lookup operations are not considered.

Detection of Replica Number

- One account holder to the first account holder administering the detection: 1
- The administering account holder to all other account holders: $k - 1$
- The other account holders to the administering account holder: $k - 1$

The sum of messages created results in: $M_{RN}(k) = 2k - 1$

Adding a New Account Holder

- Administering account holder to last account holder: 1
- Last account holder to new account holder: 1
- Response of new holder to administering holder: 1
- Transfer of account data: 1
- Confirm transfer: 1

The sum of messages created results in: $M_{Add}(k) = 5$

Detection of Correct Account Position

- One account holder to the first account holder administering the detection: 1
- Administering account holder to a random trusted peer: 1
- Steps of account position detection: number of messages is approximately the account offset x
- Response from trusted peer to administering account holder: 1

The sum of messages created results in: $M_P(k, x) = 3 + x$

Account Holder Set Locking

- The administering account holder to all other account holders: $k - 1$
- The other account holders to all other account holders: $(k - 1)(k - 1)$
- All other account holders confirm to the administrating account holder: $k - 1$

The sum of messages created results in: $M_L(k) = 2(k - 1) + (k - 1)^2$.

Account Holder Set Un-Locking

- The administering account holder to all other account holders: $k - 1$

Account Consistency

- One account holder to the account holder administering the process: 1
- The administering account holder to all other account holders: $k - 1$
- The other account holders to all other account holders: $(k - 1)(k - 1)$
- All other account holders confirm to the administrating account holder: $k - 1$

The sum of messages created results in: $M_C = 2k + (k - 1)^2 - 1$.

Account Movement

- Account holder to the new first account holder position: 1
- Forward to other new account holders: k
- From new last account holder to all other new account holders: $k - 1$
- From new first account holder to all old holders: k

The sum of messages created results in: $M_M(k) = 3k$

Graceful Account Handover

- Leaving account holder to last account holder: 1
- Last account holder to new account holder: 1
- Response of new holder to leaving holder: 1
- Transfer of account data: 1
- Confirm transfer: 1

The sum of messages created results in: $M_{GHo}(k) = 5$

5.3.2.2 Joint Mechanisms Analysis

If one of the main maintenance mechanisms (detection of replica number or detection of new account holder set) comes to the conclusion that the account holder set must be repaired, a sequence of mechanisms is executed.

Account Replica Repair

Repairing the replica number of an account holder set requires running detection of the replica number, account holder set locking, account consistency, add account holder, and account holder set unlocking. This sequence of mechanisms results in:

$M_{RR}(k) = 2(k-1)^2 + 7k$

Account Position Repair

Repairing an account holder sets position requires to run detection of correct account position, account holder set locking, account consistency, move account, account holder set unlocking. This sequence of mechanisms results in:

$M_{PR}(k, x) = 2(k-1)^2 + 8k - 1 + x$

5.4 Token-based Accounting Simulation

For simulating the token-based accounting scheme, as a basis the peer-to-peer simulator PeerfactSim.KOM (Mul, KKM+07) in version 2 was applied. PeerfactSim.KOM was developed at our institute and provides implementations of several peer-to-peer overlay networks such as Chord (SMK+01) and Kademlia (MM02) as well as an underlay model that provides an end-to-end delay for each individual connection in the overlay. Furthermore, its layered and modular architecture enables the easy expansion of the simulator for the purpose of simulating the token-based accounting scheme. An architectural overview of PeerfactSim.KOM is depicted in Figure 5.3.

In PeerfactSim.KOM in order to enable peers to communicate each peer possesses a unique identification, the GUID. A GUID has to be implemented specifically for the corresponding overlay network. Each peer implements several roles, and each role presents a distinct part of functionality the peer possesses. A specific message type is associated with each role in order to enable the simulator to deliver a message to the correct role. In each layer, multiple roles can exist. Sending messages to different roles is possible.

Measuring Traffic and Peers' Queues

Each message type provides a getSize()-method, that calculates the size of an individual message and returns it. This way the traffic created by the token-based accounting scheme can be measured.

In order to measure the download and upload traffic on a per peer basis, each peer instantiates a MessageMeterRole. For each message that is sent or received by the peer the MessageMeterRole logs the message.

The MessageMeterRole also instantiates a download queue and an upload queue. A queue calculates for each message the caused queue length, that is the time required to send or receive all Bytes in the queue. Using the queue length a peer's link load can be determined. In order to configure the queues there exist the parameters c^n_{up} and c^n_{dl} for the upload and download link capacity of peer n.

In order to evaluate the token-based accounting scheme, the overlay layer, transport layer, and network layer of the simulator were not adapted. The token-based accounting scheme was implemented into the application layer. A completely new user layer was implemented in order to simulate only behaviour relevant to the token-based accounting scheme. The details are discussed in the following section. After, the experiments simulated will be explained and finally the simulation results will be presented.

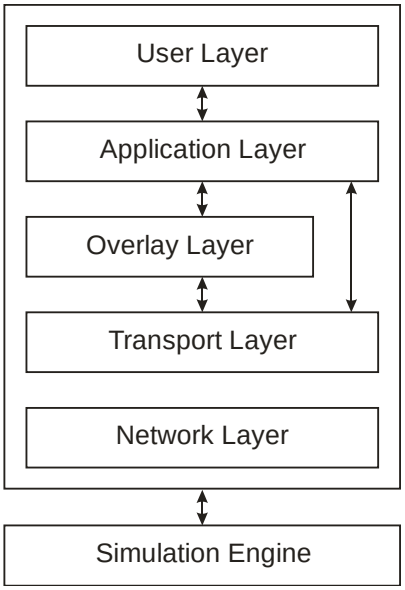


Figure 5.3: PeerfactSim.KOM: Layered architecture

5.4.1 Simulation Model

The simulation model of the token-based accounting scheme was implemented into PeerfactSim's user layer to model the users' actions and the application layer to model the token-based accounting scheme's protocols. For the overlay network, the existing Chord implementation was applied as required by the account holder set mechanisms.

Implemented Roles

The simulation model is structured according to the different building blocks of the token-based accounting scheme. The *TokenAccounterRole* implements the token-based accounting core component of the token-based accounting scheme's architecture (see Figure 3.5). The *TrustedPeerRole* implements the additional functionality required by trusted peers for issuing tokens (the TokenAggregation component within the architecture in Figure 3.5). The *AccountHolderRole* implements the protocols for checking double spending (the Aggregation Accounts sub-component of the Double Spending Detection component within the architecture in Figure 3.5). The *AccountHolderSetRole* implements all mechanisms required for maintaining an account holder set in presence of churn (the Account Holder Set Maintenance sub-component of the Double Spending Detection component within the architecture in Figure 3.5). Finally, the *TokenUserRole* implements the users' behaviour. The application scenario's functionality was also implemented here, as it was kept very simple.

In order to keep the token-based accounting simulation adaptable to different overlay networks and network layer implementations, there are two classes connecting the application layer with the overlay layer and the network layer. The *ChordFindingStrategies* implements all lookup variants required, that is, to lookup a specific GUID, find the next trusted peer starting from a specific GUID, and find a random trusted peer. Furthermore, the *ChordFindingStrategies* implements methods to return the peer's successor or predecessor, which is required for all routing processes along the ring.

5.4.1.1 TokenAccounterRole

The structure of the TokenAccounter-implementation with its core, the TokenAccounterRole is depicted in Figure 5.4.

TokenAccount

Each peer implements the TokenAccounterRole. With the TokenAccounterRole a peer has a TokenAccount storing its own and foreign tokens. The foreign tokens are stored in an account stored by the foreign peers.

Requesting Starting Tokens and a Token Aggregation

The TokenAccounterRole is responsible for requesting the starting number of own tokens from a trusted peer. The SwapAgent waits to receiving partially signed tokens and stores the new own tokens in the account when all token parts were received. The same process takes place when the peer requests a token aggregation. A token aggregation is also started by the TokenAccounterRole by sending a swap message to a random trusted peer. A random trusted peer is found by using the ChordFindingStrategies. A lookup message for a random GUID returns the GUID of the responsible peer B to the ChordFindingStrategies. Then a NextTrustedPeer message is sent to that peer B. B will check if it is a trusted peer. If it is, it will respond with its GUID to the initiating peer. If not, it will forward the message along the Chord ring to its successor. The TokenAccounterRole of B's successor will repeat this process, until a trusted peer is found. The TokenAccounterRole can now send the swap message to the found trusted peer. A SwapAgent is instantiated and will wait to receive the partially signed tokens. The signal Tokens() serves as a dispatcher to forward messages concerning tokens to the correct agent.

Transactions

For each transaction, a PayTokensAgent is responsible for sending the correct number of tokens to the provider. On the provider side, a ReceivePaymentAgent will receive the tokens. The signal Tokens() at the TokenAccounterRole is responsible for initially receiving the corresponding messages and forwarding them to the correct agent. The TokenUserRole is responsible for initiating a transaction (see below). On the service provider side, the TokenUserRole calls the provideService()-operation of the TokenAccounterRole. This operation sends a service message to the service requester. The service message contains specific information about the size of the service provided. Furthermore details about the service were omitted, as the service does not influence the operations of the token-based accounting scheme.

The requester starts the payment process with the startPayment()-operation. On the provider side the token IDs will be sent to the corresponding account holder set in order to check their validity using the sendAHUpdate()-operation. The answer is received by the signal AHSUpdateReply and forwarded to the corresponding ReceivePaymentAgent. The agent checks if sufficient tokens were accepted by the account holder set. If so, a TokenRequestMessage is sent to the requester, containing the "complete" statement. If not a sufficient number of tokens could be accepted, the provider will request further tokens using also a TokenRequestMessage containing status information about all tokens the requester sent. On the requester side, the Tokens() signal receives TokenRequestMessages and forwards them to the corresponding PayTokensAgent. If additional tokens are requested, the requested number of tokens is retrieved from the TokenAccount and send to the provider using the anotherPayment()-operation.

Furthermore Considerations

Malicious behaviour was not implemented, as this would involve the reputation system. The traffic generated from updating a peer's reputation value mainly depends on the applied reputation scheme.

5.4.1.2 TrustedPeerRole

The structure of the TrustedPeerRole and its sub classes is depicted in Figure 5.5.

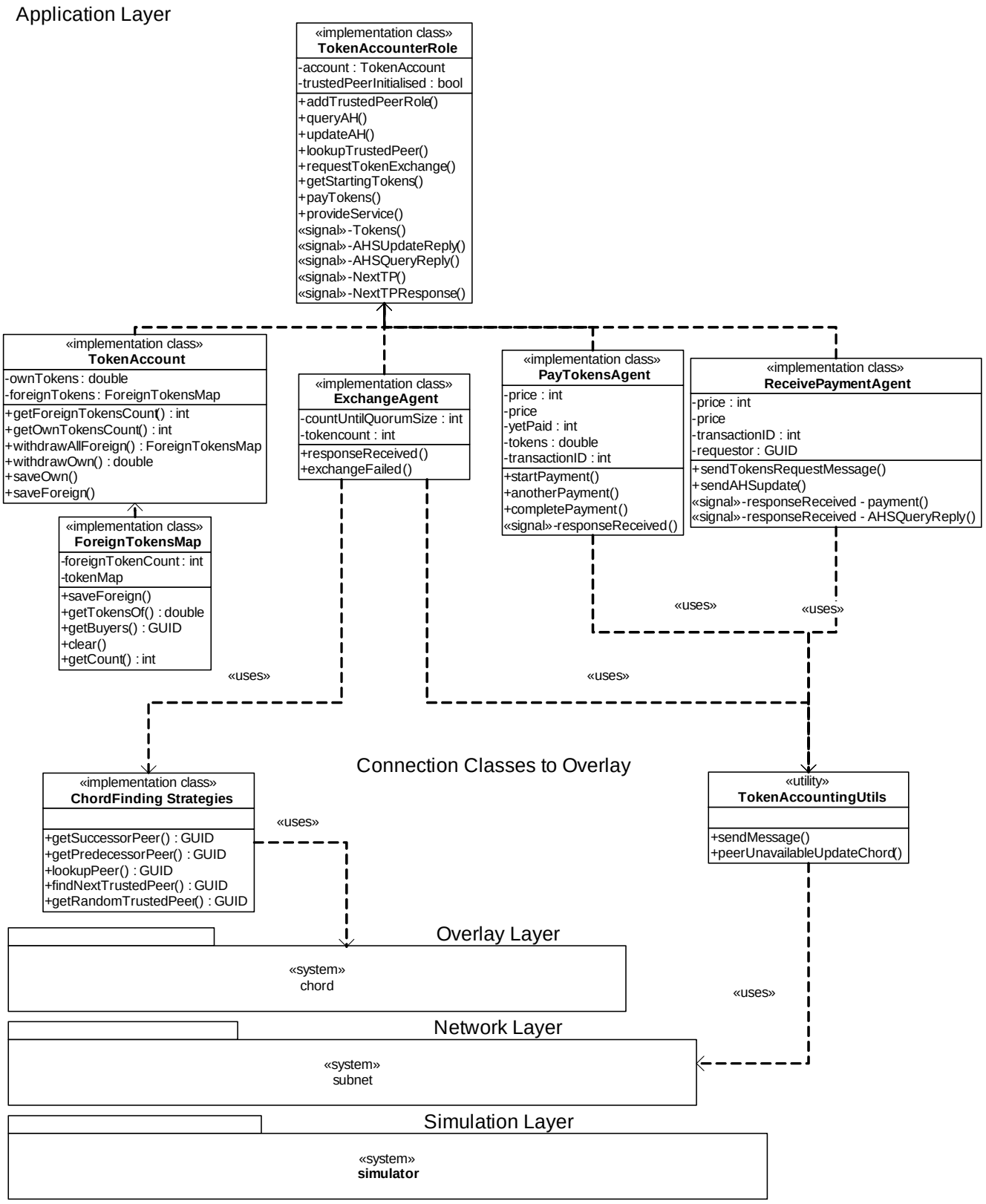


Figure 5.4: TokenAccounterRole implementation

Trusted Peer Selection

The process of determining a new trusted peer depends on the applied reputation scheme. Therefore, in the simulation of the token-based accounting scheme this process was not implemented. A simple mechanism was applied in order to select trusted peers. As a global simulation parameter the ratio p_{tp} of trusted peer to normal peers can be configured.

Whether or not a peer is a trusted peer is set by the parameter "trustedPeerInitialized" in the TokenAccounterRole. When a peer comes online, the number of online peers is determined using the central simulator's class PeerManager. It keeps track of which peers are online and which are offline. When the actual trusted peer ratio $p_{tp}^{'}$ is below the target ratio p_{tp} the peer will instantiate the trusted peer role.

Token Aggregation

When a peer SP requests starting tokens or a token aggregation, it will send this request to a random trusted peer TP_1. The trusted peer TP_1will receive this message via the signal processSwapMessage(). TP_1 will now send a request message to the SP's account holder set in order to determine the trusted peer that is to administer the token aggregation. The signal selectAggregationAdminReply() receives the response. At the selected administering trusted peer TP_{Admin} the tokens to be aggregated are received by the signal processSwapMessage(). At TP_{Admin}, a new AggregationTask is instantiated that will manage the remaining aggregation process. The AggregationTask starts with the doInitialChunk()-operation. Using the requestAHCheckTokens()-operations, TP_{Admin} will send a query message to each owner peer of the foreign tokens to be aggregated. The AccountHolderCheckRequester will collect the responses. Now the doTask()-operation is initiated. Using the lookupTPQuorum()-operation the, request message is sent to SP's account holder set. The account holder set's reply will be received by the tpQuorumFound()-signal. The number of new tokens to be created is determined and created according to the aggregation function $A(Fn_1, ..., Fn_n)$ using the newTokens()-operation. Tokens are identified by a centrally assigned tokenID. This is performed by the getFirstFreeToken()-operation in the AccountHolderMap (see below). When all these task have been accomplished, the operation doFinalChunk()-operation sends the new tokens to the quorum peers using the sendToQuorum()-operation.

At the quorum peers the signal processSwapMessage() receives the NewTokensMessage. The quorum peers will use the sendAHUpdate()-operation to update the aggregation account. Then they use the sendPartiallySignedTokensToSwappingPeer() to send the tokens to SP.

Determining Aggregation Account Positions

The DetermineAccountStartPosition class is responsible for determining the first account holder of a specific aggregation account, as described in Section 4.2.1.2. The process is started by the signal accountPosition(). The findAccountPosition()-operation performs the first lookup step. This operation also determines if another lookup step is required. This is done using the anotherLookupStep()-operation.

5.4.1.3 AccountHolerRole

The account holder set required the most complex implementation. It consists of four core parts. The AccountHolderRole is responsible for storing aggregation accounts and replying to queries and updates of aggregation accounts. The AccountHolderSetRole implements the mechanisms required to maintain an account holder set in the presence of churn. The AccountHolderSetMaintainer controls the sequence of mechanisms executed by the AccountHolderSetRole. The AccountHolderSetMap is a central class that stores the account holders for each aggregation account and controls the frequency of account maintenance operations executed by the AccountHolderSetMaintainer. Note that both the AccountHolderSetMaintainer and the AccountHolderSetMap are not roles, as they are not sending or receiving messages. The structure of the account holder set implementation is depicted in Figure 5.6.

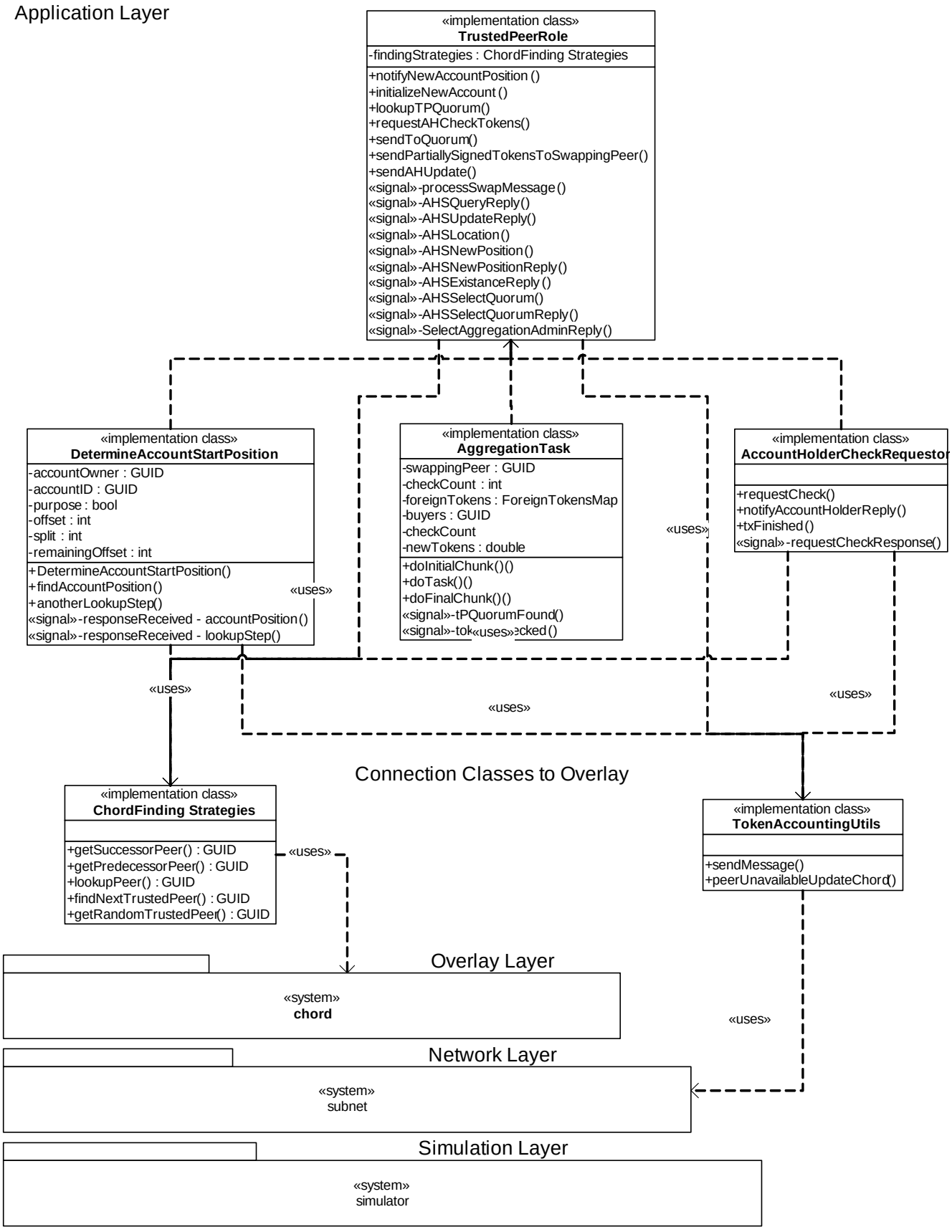


Figure 5.5: TrustedPeerRole implementation

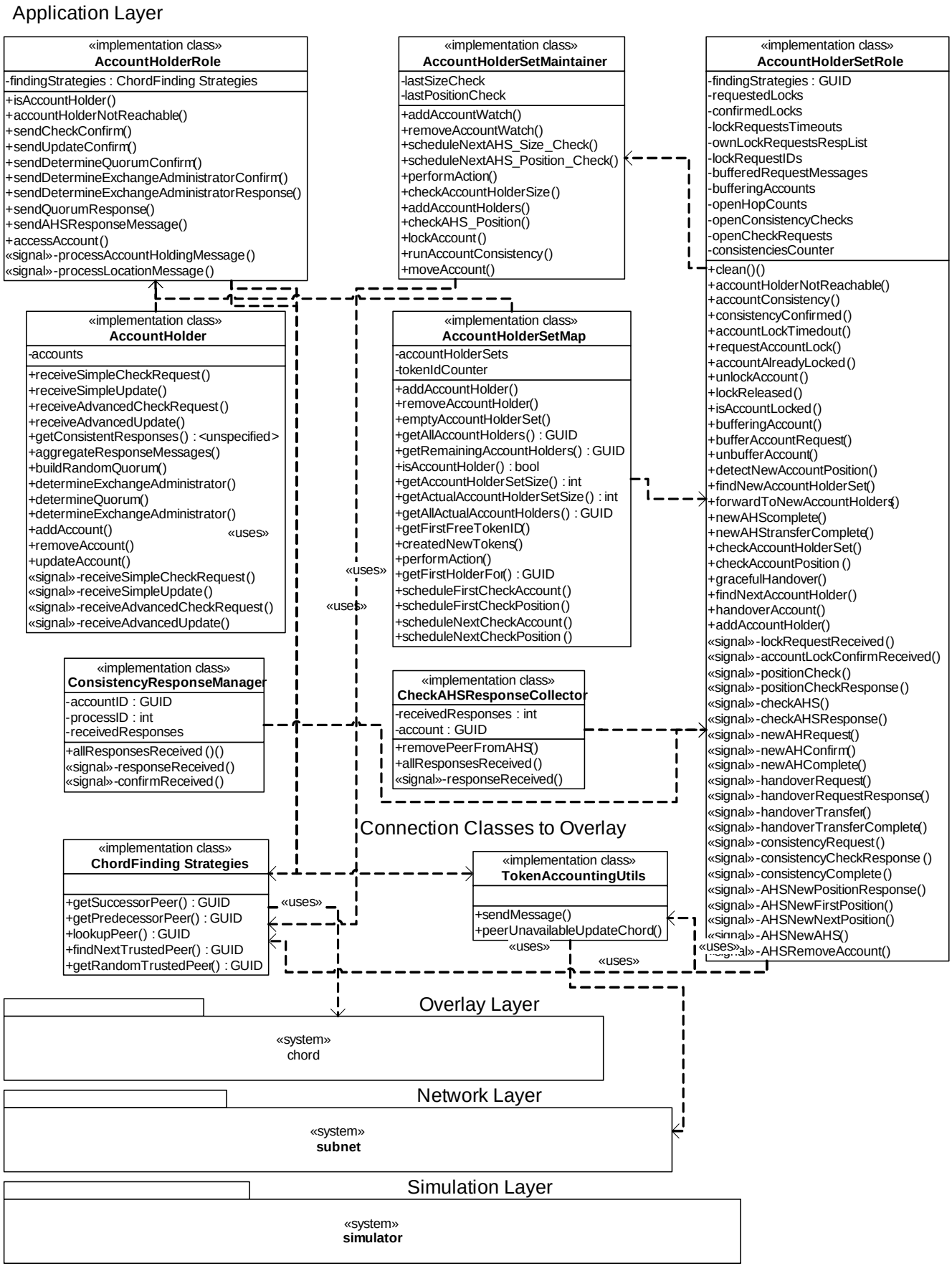


Figure 5.6: AccountHolderRole implementation

AccountHolderRole

The main responsibility of the AccountHolderRole is to forward query and update messages to the first account holder of an aggregation account and from there the send the message to the complete account holder set. This happens via the signal processAccountHoldingMessage(). It also processes messages that are used for determining the correct account position. These messages are received via the signal processLocationMessage(). The AccountHolderRole also sends response messages from individual account holders to the first account holder using the sendAHSResponseMessage(). The first holder sends the result of any query or update message to the initiating peer using the different confirm-operations.

The AccountHolder-Class is responsible for processing all queries and updates. There have been two alternatives for handling tokens in the simulation implemented. The simple query and simple update do not check individual tokens. They simply compare if the number of tokens are consistent. The advanced query and update method compare each token by token ID and status. For each token an integer value as status is administered. The status gives information if a token is unused and from how many quorum peers a token update about this new token was received. Furthermore, a “planned to spend” and “spent” status exist. This way double spending etc. can be detected by an account holder. In both alternatives a token is removed from an aggregation account when it was swapped by a foreign peer. The simple alternative was implemented in order to speed up the simulation.

Furthermore the AccountHolder-Class of the first account holder is responsible for collecting the response messages of the remaining account holders and checking the consistency of these messages. If more than 50% of the messages to be received are consistent, the AccountHolder triggers the AccountHolderRole to send a confirm message.

Two query types are processed in a special manner for reasons of expediency. The queries for determining an aggregation administrator and for determining a quorum are only processed at the first account holder; the remaining account holders deliver meaningless results in the first holder. Simulating the real processes would require calculations, but no further message transfer. Therefore, these simplifications were chosen. In theses queries, the first holder will lookup a random trusted peer as administrator responsible for a quorum of random peers when it has received more than 50% of the possible response messages from the quorum peers. Then the first account holder delivers the result to the query initiator.

AccountHolderSetRole

The account holder set role implements the mechanisms for maintaining the account holder sets as discussed in Section 4.2. Similar to the account query and update processes, the consistency mechanism also supports two alternatives. The simple version checks only the consistency of the number of tokens in each account holder’s aggregation account. The advanced version, also checks the consistency of the tokens’ stati. In the advanced version the token’s status held in the majority of peers is set as the new token status. The simple version builds the resulting number of tokens based on simple majority.

For two mechanisms separate classes are instantiated when they are executed.

The CheckAHSResponseCollector collects all the responses of a checkAccountHolderSize()-operation started at the AccountHolderSetMaintainer. When the collector has received responses from all account holders of the corresponding set or a timeout occurs, the actual account holder set is determined from the responses and distributed to all account holders. Peers that responded, but are no longer account holders for that account are informed in order to remove that aggregation account.

The ConsistencyReponseManager is instantiated at each peer of an account holder set for which a runAccountConsistency()-operation was started by the AccountHolderSetMaintainer. Each account holder collects all responses from the other holders. When all responses are received or a timeout occurs, the responses are compared and the resulting account information is updated and distributed (see above).

AccountHolderSetMaintainer

AccountHolderSetMaintainer accesses the AccountHolderRole in order to perform the account holder set maintenance processes. The two main operations are checkAccountHolderSize() and checkAHS_Position(). The first will determine the actual account holder set size. If the account holder set is short in size, the account will be locked using the lockAccount()-operation and then account consistency is achieved using the runAccountConsistency()-operation. Now, the addAccountHolder()-operation is executed as many times as account holders are missing in the account. Finally the account will be unlocked. The checkAHS_Position()-operation will determine the optimal position of the first holder of an aggregation account. If this position deviates from the actual position by more than two hops, the account is locked, consistency achieved, the account moved using the moveAccount()-operation, and finally the account is unlocked.

AccountHolderSetMap

The AccountHolderSetMap was created in order to consolidate information that normally would be stored with account holder sets redundantly: This way the simulation saves memory.

The AccountHolderSetMap stores for each peer its account holder. Each time an account holder detects a change in an account holder set, it informs all other account holders of this set about it. Therefore, the AccountHolderSetMap can be applied to model the status of account holder sets. The account holder map is a current view, and might deviate from the actual situation in the simulation i.e., an absent account holder will not be reflected in the AccountHolderSetMap until another account holder detects this.

The AccountHolderSetMap controls when checkAccountHolderSize()-operations and checkAHS_Position()-operations are executed for an account. A random account holder of an account holder set is chosen to start the maintenance operation. This models the decentralised monitoring in an account holder set the last time a specific operation was executed. This implementation also saved memory in the simulation.

5.4.1.4 TokenUserRole

The tasks of the TokenUserRole are twofold. The TokenUserRole implements the user's behaviour as well as the application that the token-based accounting scheme accounts for.

Application Model

As the application, a file sharing scenario was selected where peers request complete files from each other. In this scenario, searching for a specific service and the kind of service delivery do not influence the token-based accounting scheme. Therefore, the file sharing service can be modelled in a simple way, so search for service, as well as chunking files are not considered in the simulation. Furthermore, individual files and their distribution in the network are not simulated, as the distribution of files in the system is not in focus of the evaluation.

In the resulting simulation model, each file has a specific size in MBytes. For each MBytes file size s the requester peer has to pay one token. The service delivery is simulated by sending one message of the specific file size as payload. The file size s_i for transaction i is selected at the time of the service request using a probability distribution s. Also, a random online peer is selected as service provider. Note, the file size distribution s also corresponds to the transaction value.

Application Operations

When a user requests a service, it sends a serviceRequestMessage to the selected service provider. Their message is received via the serviceRequestReceived()-signal. If the service provider is available for transactions (see below) it will accept the service request; otherwise it will deny it. At the requester the

signal requestAcceptanceReceived()-signal receives the provider decision. In case of denial, the service requester selects another peer. In case of acceptance the acceptance message contains the price of the service. The requester peer will request the service if it has sufficient number of own tokens available. It will then call the payTokens()-operation of the TokenAccounterRole; if not, it will deny the service. The final requester's decision is received at the provider via the providerDecisionReceived()-signal. The provider will call the provideService()-operation at the TokenAccounterRole in order to instantiate a ReceivePaymentAgent, and to provide the service.

In the case the requester does not have sufficient tokens for the transactions, the peer will swap its foreign tokens using the swapTokens()-operation - if the peer has more foreign tokens then the lower swap threshold b_{min}. A new transaction will be performed when the aggregation is completed. Otherwise, another transaction will be scheduled.

User Behaviour

The first time a user joins the system it will request a starting number of tokens b_{start}. b_{start} is an application dependent parameter. If no aggregation account exists yet for this peer, it will be created by the trusted peer before the starting tokens are issued using the initializeNewAccount()-Operation of the TrustedPeerRole.

When a peer receives its starting tokens it will start to request services. Using a probability function, the time of the next transaction is determined and the transaction is scheduled using the scheduleNextTransaction()-operation. After the transaction finished, the next transaction is scheduled. If after a peer provided a transaction the swap threshold b of foreign tokens is exceeded, a peer will swap all its foreign tokens for new tokens using the swapTokens()-operation.

Churn

When churn is simulated there are several parameters to be determined. The peers' lifetime distribution and offline-time distribution are the core parameters to be defined. Furthermore, a user can log off from the system or can simply disappear from the system. When a peer logs off, it will transfer its aggregation accounts to another peer using the gracefulHandover()-operation of the AccountHolderSetMaintainer. When a peer joins the system, the lifetime is determined and whether the peer will log off or leave. Accordingly, a log off or an offline will be scheduled using either the scheduleLogOff() or a scheduleOffline()-operation. The decision if a peer logs off depends on the probability $p_{log-off}$. When a peer leaves the system, the peer's offline time is determined and the next join is scheduled using the scheduleOnline()-operation.

For each peer, a period before leaving the system is determined when peers will not get involved in any transactions. This period is referred to as "dead-period-before-leave". The dead-period-before-leave simulates how when a user closes a peer-to-peer application, it will not close until all current transactions it is involved are finished. This period typically lasts some seconds. During this period, a peer will not accept transactions as provider, to become an account holder, or to administer a token aggregation or an account holder set maintenance operation.

5.4.2 Experiments

Simulated is the file sharing scenario described above. 5.4 summarises the simulation parameters. The core interests of the simulation is the influence of the quorum size t, the account holder set size k, and of churn on the traffic created by the token based accounting scheme.

In order to analyse the influence if these parameters on the created traffic, the measurements for four different parameter values will be analysed. Using four measuring points, it can be analysed whether the created traffic develops in a linear way or in different way. In order to analyse the effects of one

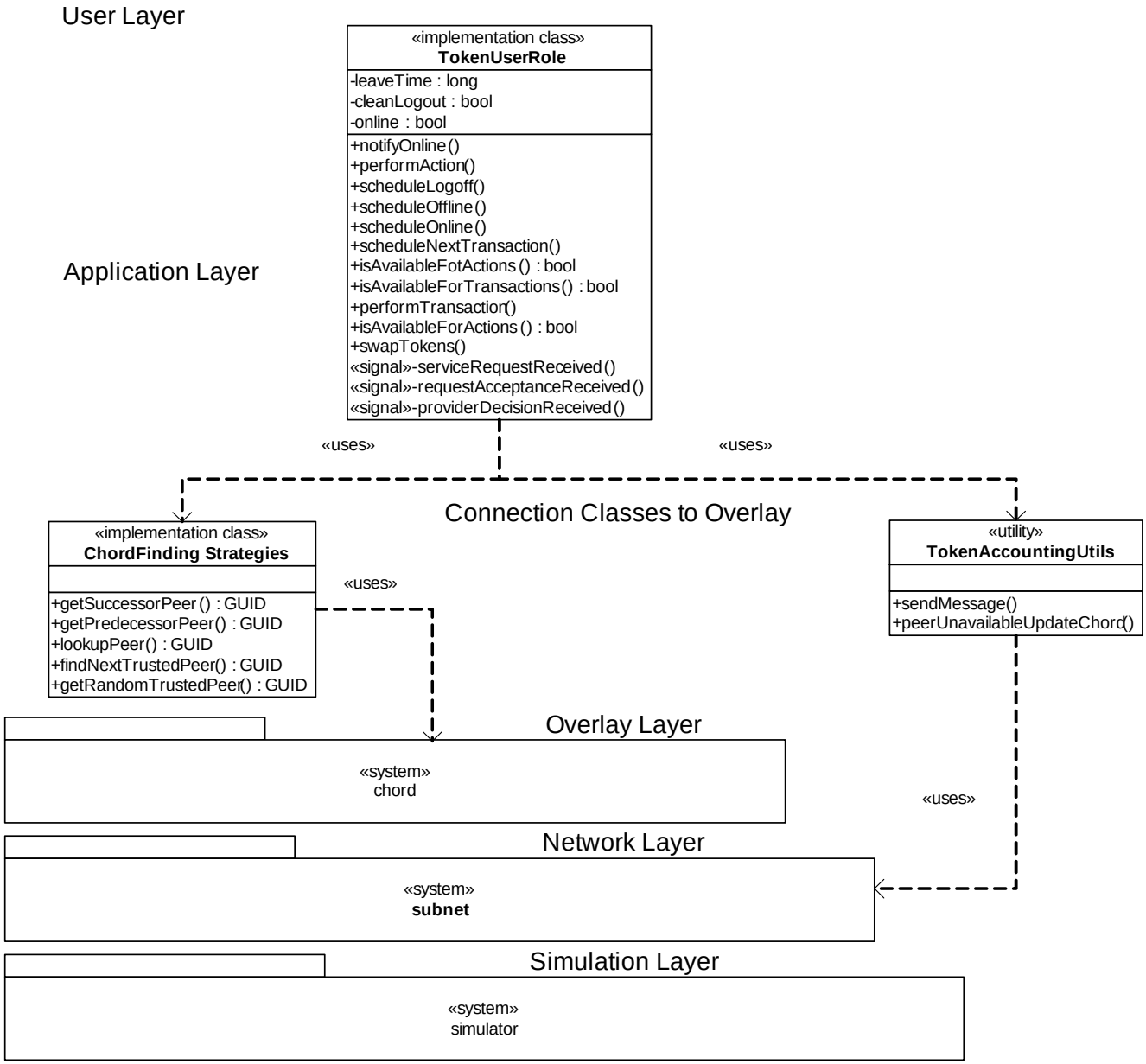


Figure 5.7: TokenUserRole implementation

parameter on the token-based accounting scheme, a base scenario is defined and the experiments will vary only one parameter at a time (ceteris paribus).

Now the base scenario is described.

5.4.2.1 Experiment Parameter Overview

Basic Settings

- *Duration*: The selected scenario will be simulated for the time of one day (24 hours).
- *System Size* (N): The system size N is 1000 peers. A larger system size led to a simulation that exceeded the selected duration.
- *Used Overlay*: For the peer-to-peer system the Chord implementation in PeerfactSim.KOM was applied.
- *Peer Bandwidth* (c): All peers have the same bandwidth available. A typical DSL-link with 128 kBit upstream ($c^i_{up} = c_{up} = 128\,kbit$) and 1024 kBit downstream ($c^i_{dl} = c_{dl} = 1024\,kbit$) was selected.
- *Key Length* (r): The key length of the peers' key and the system key pair. Since the hypothetical TWIRL machine can factorise RSA-keys within one year (ST03), today a length of 2048 bit is suggested. Although the BLS-threshold scheme uses shorter signatures, as a worst case decision in the simulation r was set to 2048 in order to reflect the traffic that the use of the URSA-scheme would introduce.
- *Trusted Peer Ratio* (p_{tp}): The trusted peer ratio was set to $p_{tp} = 33.33\%$. The selected churn (see below) means that approximately 30% of all peers are concurrently online. In order to enable a varying quorum for the largest quorum size $t = 17$, a total number of $T = 100$ trusted peers seemed a reasonable base value. Accordingly, p_{tp} was selected.
- *Trusted Peers' Trustworthiness* (p_g): It is assumed that the ratio of good peers among the trusted peers is at least $p_g = 50\%$. This value is required in order to calculate the required quorum size.
- *Account Holder Set Maintenance* (f_{ck}, f_{cp}): The account holder set methods depend on the frequency the account holder set size is checked f_{ck} and the frequency the account position is checked f_{cp}. For the simulation f_{ck} is set to 300 seconds and f_{cp} is set to 600 seconds, likewise for the simulation for determining the optimal account holder set size.

Quorum Size

For the quorum size, different performance values of the token-based accounting scheme are applied. The scheme is evaluated with Trust Levels L = 99%, 99, 9%, 99,99% and 99,999%. This results for the selected $p_g = 50\%$ in quorum sizes t = 7, 10, 14, and 17.

Account Holder Set Size

Because the preferred account holder set size k's influence of different account holder set sizes on the created traffic, the account holder set size will be varied k = 6, 8, 10, and 12, as these are the results from the simulation of the account holder set maintenance mechanisms (see above).

Churn

Churn is modelled using the results from (SENB07b).

- *Online Time* (d_{on}): The base peer lifetime is a Weibull distribution with $\alpha = 0,61511$ and $\beta = 169,5385$, measured in minutes. For the simulation, we assumed that peers are online at least

$d_{on,\,min} = 30$ seconds. This assumption helped to keep the Chord ring more stable. Also, as DSL-connections are typically reset after 24 hours, the maximum possible online time was set to $d_{on,\,max} = 24$ hours. The maximum simulation time was 24 hours. A longer online time could not be reflected in the simulations.

- *Offline Time* (d_{off}): The base peer offline distribution is a Weibull distribution with $\alpha = 0,47648$ and $\beta = 413,6765$, also measured in minutes. The same minimum $d_{off,\,min} = 30$ seconds and maximum $d_{off,\,max} = 24$ hours have been applied to the offline time distribution.

In order to analyse churn, both distributions are multiplied with factors d_{var} of 1.0, 0.5, 0.33, 0.25.

At the beginning of the simulation ,all peers are offline. In order to bring the peers into the system, each peer will join after half of an offline time period. That simulates that the system was pre-existing.

- *Log Off Behaviour* ($p_{log-off}$): The basic probability for a log off (in contrast to a leave) was set to $p_{log-off} = 90\%$. As users typically close the application when they leave the system, the remaining 10% are thought to reflect crashes, connection failures, etc.

Application Parameters

Note that the application parameters are required to create traffic for the token-based accounting scheme. It is not used to evaluate the efficiency of a file sharing application. This is beyond the scope of this dissertation.

Moreover, there are no measurements of the transaction frequency and file size distributions for the scenarios where the churn was measured. As it is more important for the simulation of the account holder set to use a realistic churn model in the simulation, it was decided to apply a realistic churn model, and create distributions for transaction frequency that model meaningful user behaviour in a scenario where the token-based accounting scheme is applied.

- *Transaction Frequency* (f_t): A peer will request services according to a normal distribution f_t with a mean $\mu = 1800$ seconds (30 min) and a standard deviation $\sigma = 1800$. The distributions minimum was set to 60 seconds and maximum to 7200 seconds (2 hours).

- *File Size and Transaction Value* (s): The distribution of file sizes is thought to model an mp3-file sharing scenario. Accordingly, a normal distribution was applied with $\mu = 6$ and $\sigma = 6$. Also for this distribution a minimum of $s_{min} = 1$ and a maximum of $s_{max} = 13$ was set in order to reflect typical file sizes for such an application type in MBytes.

- *Dead Period Before Leave* (d_{dead}): When closing an application, the time that is required to finish all open tasks. This period was set to 20 seconds.

- Aggregation Function $M = A(Fn_1, ..., Fn_n)$: In the file sharing application a peer receives as many new tokens as old tokens were aggregated. $M = n$, where n is the number of foreign tokens sent for aggregation.

Token-Accounting User Behaviour

- *Swap Threshold* (b): User exchanged all their foreign tokens when they collected more then 25 foreign tokens.

- *Lower Swap Threshold* (b_{min}): When a user is in need for tokens in order to do further transactions, it collects at least 3 foreign tokens before swapping them. This is also the minimum value of a transaction.

- *Starting Tokens* (b_{start}): At the beginning of the simulation each user receives 50 own tokens.

Table 5.4: Simulation Scenario Parameters

Parameter	Description	Values
N	Total number of peers in the system. All peers in the system are normal peers.	1000
c_{up}^{i}	Upload link capacity of peer i	128 kBit
c_{dl}^{i}	Download link capacity of peer i	1024 kBit
r	Length of used keys	2048 bit
p_{tp}	Target ratio of trusted peers to normal peer in the system	33,33%
T	Total number of trusted peers in the system; results from p_{tp}.	≈ 300
t	Quorum size	17, 14, 10, 7
L	Trust Level, depending on T, t, p_g	99%, 99.9%, 99.99%, 99.999%
k	Preferred Account Holder Set Size	6, 8, 10, 12
x	Account Shift Value	5
f_{ck}	Frequency account holder set size check	300 sec
f_{cp}	Frequency account holder set position check	600 sec
f_t	Frequency a user requests a service	NormDist(1800; 1800), min = 60 sec, max = 7200 sec
s	File size distribution resp. service value distribution	NormDist(6; 6), min = 3, max = 13
b	Swap threshold	25
b_{min}	Lower swap threshold	3
b_{start}	Number of starting tokens	50
$A(Fn_1, ..., Fn_n)$	Aggregation function	$A = n$
d_{on}	Peers' lifetime distribution	Weibull(0,61511; 169,5385); min = 30 sec, max = 86400 sec
d_{off}	Peers' offline time distribution	Weibull(0,47648; 413,6765); min = 30 sec, max = 86400 sec
d_{var}	Churn modification factor	1
$p_{log-off}$	Probability for logoff	90%
d_{dead}	Dead period before a peer leaves	20 sec
V	System-wide traffic volume	
v	Traffic volume per peer	

Table 5.5: Simulation Message Size Parameters

Parameter	Parameter	Size [Byte]
MTU	l_{MTU}	1500
Overlay Address	l_{ID}	20
I²P Header	l_{IP}	10
Transport Header	l_{TCP}	10
Key length	r	256
Delimiter	l_{del}	2
Response Identifier	l_{resp}	4
int value	l_{int}	4
long value	l_{long}	8
double value	l_{double}	8

Token	Parameter	Size [Byte]
Fresh unsigned	U	282
Partially Signed	P	540
New Own Token	T	540
Used Foreign Token	F	1096

5.4.2.2 Message Sizes

Calculating Message Sizes

Each message sent in the token-based accounting simulation has a specific size. The size is composed of the basic required headers of the IP- protocol and the TCP or UDP-protocol, as well as the information the message carries. Each information field is delimited by a delimiter. All message types can carry different amount of information. Depending on the information carried, a message's size is calculated in the simulation. Table 5.5 describes the assumed sizes of different types of information.

The message sizes are considered fixed parameters in the simulation and result from either the applied overlay network or the messages used in the token-based accounting scheme. They will not be varied in the simulation.

The MTU is used in order to calculate the number of fragments a message is split into, and with this to calculate the actual traffic sending a specific message creates because per fragment IP- and TCP/UDP headers are required to be added.

Token Sizes

There are four different kind of tokens used inside the token-based accounting scheme. When tokens are created in the first place they contain the information required to identify it. These fresh unsigned tokens are composed of a date (type: long), a serial number (type: int), an owner ID (type: key), an account ID (type: int), and the required five delimiters. This results in 282 Bytes. Partially signed tokens also contain a signature (type: key), as well as a further delimiter. This results in 540 Bytes. New own tokens have the same size. Used tokens additionally contain transaction ID (type: int), transaction time (type: long), transaction object (type: overlay ID), provider ID (type: key), owner signature (type: key), and six delimiters. This results in a size of 1096 Bytes. Furthermore accounting information added to used tokens was not considered in the simulation.

5.4.2.3 Experiment Overview

This section gives an overview on the performed experiments and the base experiment the other experiments are derived from.

Performed Experiments

The basic experiment uses the values given in Table 5.4 for the parameters of the token-based accounting scheme. All experiments use as traffic parameters the values given in Table 5.5. Three main influencing factors on traffic generated by the token-based accounting scheme will be analysed: The

Table 5.6: Experiments Overview

	t	k	d_{var}
Base	17	10	1
Quorum	14		
	10		
	7		
AHSS		12	
		8	
		6	
Churn			$\frac{1}{2}$
			$\frac{1}{3}$
			$\frac{1}{4}$

influence of the quorum size t, the influence of the account holder set size k, and the influence of churn by varying the churn modifier d_{var}. For each factor, four different scenarios were simulated; this allows for further understanding of how traffic develops; i.e., is the traffic growth linear, exponential, etc.

In order to analyse the influence of the main parameter of the token-based accounting scheme on the created traffic, one parameter at a time is varied and all other parameters are kept constant (ceteris paribus).

The influence of the quorum size t on the traffic created is analysed by using different Trust Levels $L(T, t, p_g) = \{99\%, 99.9\%, 99.99\%, 99.999\%\}$. It is assumed that at least $p_g = 50\%$ of the trusted peers act honestly. For the system size of $N = 1000$ peers and a trusted peer ratio $p_{tp} > 10\%$, the required quorum sizes result in $t = \{7, 10, 14, 17\}$.

The account holder set size k depends on the churn in the system. Accordingly, the account holder set would be increased with stronger churn. However, in order to be able to analyse the influence of the account holder set size k on the traffic generated, churn has to be kept constant. Otherwise it would interfere with the results. The simulated values are deduced from the simulations of the account holder sets, presented in Section 5.2.2. Set sizes of $k = \{6, 8, 10, 12\}$ will be simulated in order to cover a broad range of churn scenarios.

In order to analyse the influence of different churn, a basic churn model is used and this is modified by the churn factor d_{var}. In the basic churn model the results from (SENB07b) are applied, which have also been applied for the account holder set size simulations (see Section 5.2.2). Both, peer lifetime and peer offline time, are distributed according to Weibull distributions with parameters listed in Table 5.1. Likewise, the analysis of the account holder set size, both distributions are modified with the churn factor $0 < d_{var} \leq 1$, which increases the influence of churn, as peer lifetime and peer offline time are reduced. d_{var} is chosen as $d_{var} = \left\{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}\right\}$.

Furthermore, in order to study the scalability of the token-based accounting scheme, i.e. the effect of different system sizes, the system sizes of $N = \{1000, 250\}$ are used.

Base Experiment

The base experiment uses the highest trust value $L(T, t, p_g) = 99.999\%$, which results in a quorum size of $t = 17$. The base account holder set size is set to $k = 10$, in order to allow conduct the churn experiments ceteris paribus. As base churn the churn factor is set to $d_{var} = 1$.

Table 5.6 summarises the performed experiments.

Number of Repetitions per Experiment

In order to determine the required number of repetitions per experiment, the central limit theorem (BGG94) is applied. Each experiment runs with either 1000 or 250 peers. Thus, according to the the

central limit theorem, it can be assumed that the average traffic created per time period is normally distributed. For computing meaningful confidence intervals, the sample size should be at least ten. Therefore, each experiment is repeated 12 times.

As the traffic is normally distributed for computing the confidence intervals, the student t-distribution has to be applied for the given experiment size (cf. (BGG94)).

5.4.3 Simulation Results On Transaction Traffic

5.4.3.1 Graph Configuration

System-wide Traffic Graphs

Each data point in the graphs represents a 20 minute time period in the simulation. For each data point, the average amount of traffic generated in the complete system during one minute of simulation time is plotted. Twenty minute plot intervals were chosen, as with smaller time intervals error bars could not clearly be depicted. The error bars represent the 90% confidence interval.

System-wide Traffic Ratio Graphs

The system-wide traffic ratio graphs represent the same amount of traffic like the system-wide traffic graphs. Also, the error bars represent 90% confidence intervals.

There are two types of ratios computed. The first ratio type is the ratio of the traffic generated by a specific mechanism divided by the total amount of traffic generated in the experiment. These ratios allow understanding the cost of using the token-based accounting scheme in a specific usage scenario. The graphs show if usage bears prohibitive high costs or not.

The second ratio type is the ratio of the traffic generated by a specific mechanism of the token-based accounting scheme divided by the complete traffic generated by the token-based accounting scheme. These ratios show which mechanisms have the strongest influence on the traffic generated by the token-based accounting scheme.

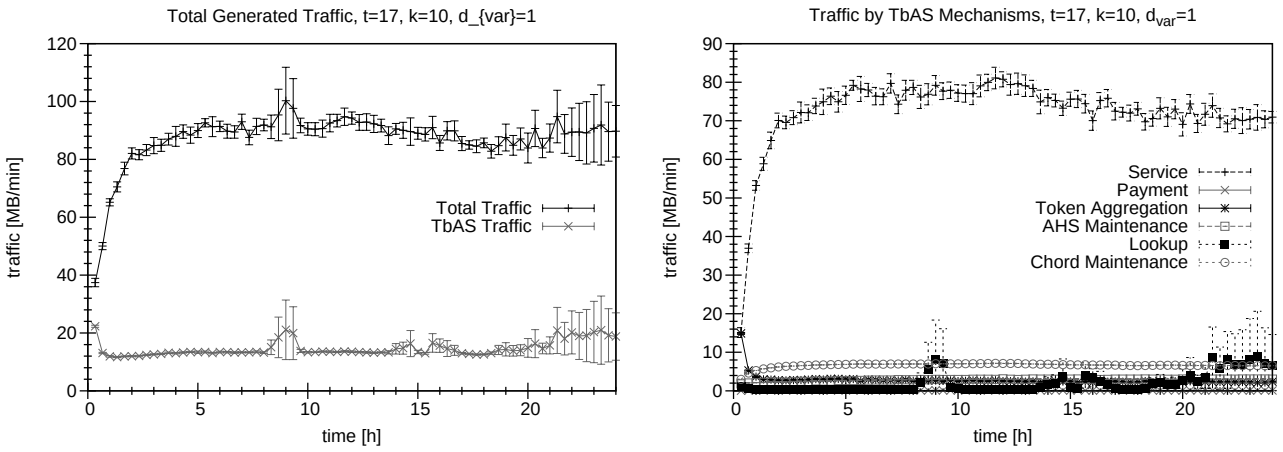
5.4.3.2 Base Experiment Result

The traffic generated in the base experiment with a quorum size of $t = 17$, an account holder set size $k = 10$, and a churn factor of $d_{var} = 1$ is depicted in Figures 5.8, 5.9, and 5.10. In the beginning of the simulations peers join the system the first time distributed with half of the offline time distribution. The average join time is 1.5 hours; however, as the join time is Weibull distributed the majority of peers join up to 1.5 hours. Afterwards the joins diminish. As the first step when going online for the first time, each peer requests its starting tokens. Therefore, at this time also the peer's aggregation account has to be created. When both tasks were completed, the peer schedules its first transaction.

In this section the figures show per data point the average upload traffic per minute generated over a 20 minute time period. The error bars represent the 90% confidence interval. Within each time period the number of active peers was recorded in order to determine the average traffic per peer. A peer is active within a period if it sent a message.

Overall Traffic Observations

Figure 5.8 a) shows the complete system-wide traffic versus the system-wide token-based accounting traffic. At first the complete traffic increases sharply. This is due to the increasing number of online peers that start to consume services. The token-based accounting traffic is highest in the beginning with 22.07 MBytes per minute. This is due to the newly joining peers that request the starting tokens and get an aggregation account initialised. Then token-based accounting traffic decreases, because the



(a) Complete traffic vs. accounting traffic

(b) Traffic breakdown

Figure 5.8: System-wide traffic for $t = 17$, $k = 10$, $d_{var} = 1$

number of new joins per minute also decreases. The token-based accounting traffic then levels off at approximately 13.4 MBytes per minute, which is per active peer on average approximately 30.31 kBytes per minute. After service traffic levelled, approximately 75 MBytes service traffic per minute is created. The ratio of service traffic to token-based accounting traffic is approximately 5.6 in the simulated file sharing scenario.

The token-based accounting traffic is flat most of the time. However, in between there are traffic increases. Figure 5.8 b) shows the explanation for this increased traffic. The reason is an increase in lookup traffic. The Chord overlay causes this. Lookup messages have a time-to-live of 100 hops. When the Chord-ring has a falsely set successor, a message might be rooted to a node that is not responsible for a key, because it knows a predecessor that should be responsible for it. Then the message is rooted again along the ring. If the falsely set successor is not repaired, the lookup message travels its 100 hops before it is discarded. In such a situation in the token-based accounting scheme, timeouts become active and repeat the lookup four times. Accordingly, instead of one lookup message travelling a maximum of ten hops in a system of 1,000 nodes, this lookup message travels now 500 hops in total also considering the repeated lookups. Accordingly, the lookup traffic increases by more than a factor of 50, because on average, a lookup message travels less then ten hops. When the Chord ring operates correctly, lookups create on average approximately 0.4 MBytes of traffic per minute. The peak average traffic is 15.47 MBytes per minute. This is an increase by a factor of almost 40. Furthermore, it must be considered that this traffic has to be shared among a small number of peers, as routing around the ring requires $\log(N)$ at maximum, which is ten peers in the simulations. Obviously, this is a malfunction of the implemented Chord protocol. This deviation from normal operation will not be considered in the further analysis, as for a real world deployment an improved version of Chord or a different overlay network would be employed.

Traffic Breakdown Observations

Figure 5.9 shows the breakdown of the traffic into the main groups of the token-based accounting scheme - Payment, Token Aggregation, Account Holder Set Maintenance, Lookup, and Chord Overlay Maintenance. Figure 5.9 a) shows the system-wide traffic, Figure 5.9 b) shows the average traffic per active peer.

Analysing this traffic breakdown shows that Chord maintenance creates the largest amount of traffic, not considering the lookup traffic in the presence of failures in the Chord ring. After the join process, Chord maintenance traffic is almost constant with approximately 6.8 MBytes per minute of system-wide

traffic on average. For an active peer this relates to 15.95 kBytes traffic per minute on average. Not considering the service provisioning traffic, this is about 52% of the overall traffic (see Figure 5.10 b).

Lookup traffic is the traffic that stems from lookups done on peers and trusted peers by the token-based accounting scheme. System-wide lookup traffic is approximately 0.4 MBytes per minute on average, when the Chord ring is working correctly; this is minor compared to Chord maintenance traffic. Per active peer, this is on average 0.91 kBytes per minute (see Figure 5.9 b)). The proportion of lookup traffic within all token-based accounting traffic is approximately 3%.

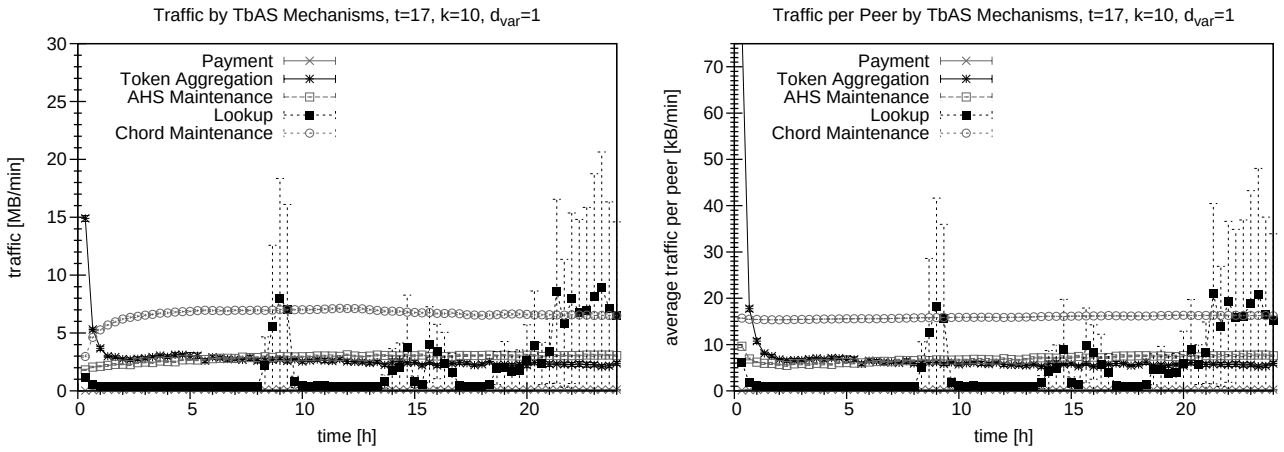
Maintenance of the account holder sets generates the largest proportion of traffic among the token-based accounting mechanisms with approximately 2.95 MBytes system-wide traffic per minute on average. Per active peer this is on average 6.96 kBytes per minutes. Figure 5.10 shows that the maintenance mechanisms consume 21.71% of the generated traffic, not considering service traffic.

The largest percentage of traffic is created by token aggregation. This traffic is highest in the beginning due to the large number of peers requesting their starting tokens. Figure 5.11 b) shows that in the first 20 minute time period on, average 296.15 token aggregation processes are completed per minute, which have a total value of 14592.46 tokens on average (see Figure 5.11 a)). This creates a system-wide traffic of 14.75 MBytes on average during the first 20 minutes of the simulation. This is 79.22 kBytes per active peer per minute on average, and proportion of 66.83% of all token-based accounting traffic. After this start-up phase, aggregation traffic is decreases slightly over time due to fewer aggregations (see Figure 5.11 b)). The maximum system-wide traffic after the start-up time is 3.12 MBytes per minute on average. This relates to 6.63 kBytes per minute per active peer. After 24 hours simulated time system-wide token aggregation traffic is 2.31 MBytes per minute on average; this relates to 5.40 MBytes per peer per minute. Not considering service provisioning traffic after the start-up phase token aggregation consumes on average a proportion between 21.89% and 15.75% of all token-based accounting traffic. Observing the traffic generated per token aggregation process results in 541.70 kBytes on average; at the beginning of the simulation the token aggregation processes are more expensive, because this includes the creation of aggregation account at the account holders. Here, a maximum traffic of 1019.91 kBytes per token aggregation process can be observed.

Payment creates the smallest amount of traffic; at the beginning of the simulations, there are less services requested and paid for, because few peers joined the system. The maximum payment traffic is approximately 0.12 MBytes per minute of system-wide traffic on average. This is approximately 0.29 kBytes per active peer per minute on average. Like token aggregation traffic, payment traffic also decreases slightly over time due to a slight decrease in the number of transactions performed (see Figure 5.11 b)). After 24 hours simulated time, system-wide payment traffic is 0.11 MBytes per minute on average. This is 0.26 kBytes per peer per minute on average. On average payment consumes a proportion of 0.83% of traffic among the token-based accounting traffic. Payment traffic per transaction is constant over the simulation time- approximately 5.55 kBytes per transaction.

Both the payment and the token aggregation protocol require access to aggregation account. One query and response of an aggregation account consume 76.45 kBytes.

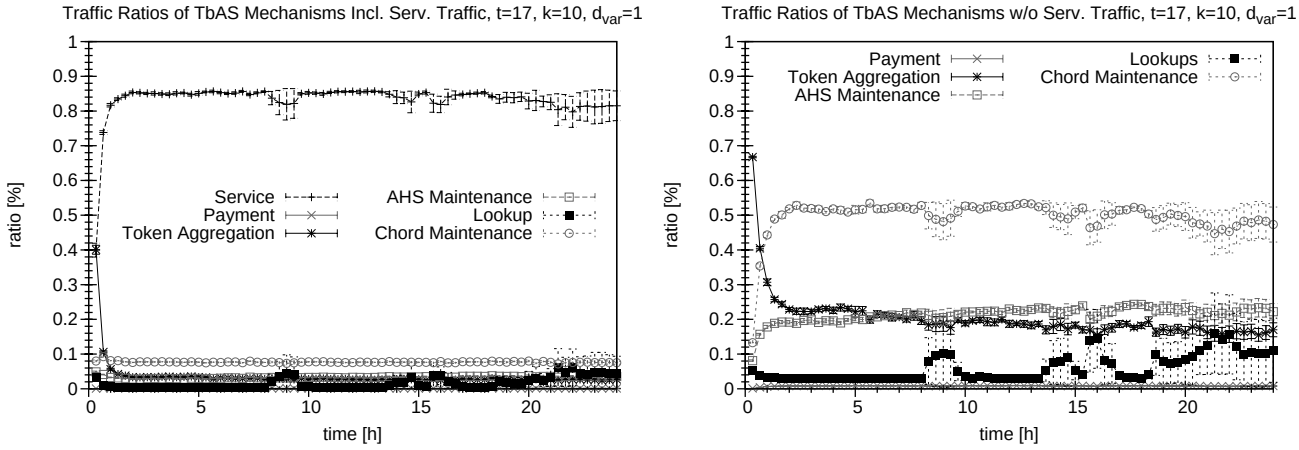
In summary, there is payment traffic and token aggregation that depends on the number of transaction performed in the system. Maintenance traffic is constant per transaction as well as Chord maintenance traffic. Lookups are mainly used within payment and token aggregation processes. Also, lookups are required for determining a new position of an account holder set within maintenance. Analysing the purpose of lookups, 41.65% are used within maintenance mechanisms, and 58.35% are used within payment and token aggregation. Not considering the lookups due to malfunction of Chord described above, on average one transaction creates approximately 293.58 kBytes of traffic, which is distributed over the peers participating in the transaction and the corresponding token aggregation process; this is at least 20 peers for transaction and aggregation, plus 20 account holders. This traffic is distributed unevenly among the peers. In the scenario this is 56.22 kBytes of variable traffic per peer per minute on average. Constant traffic due to account holder set maintenance and Chord maintenance is approx-



(a) System-wide traffic

(b) Average traffic per active peer

Figure 5.9: System-wide token-based accounting traffic breakdown for $t = 17$, $k = 10$, $d_{var} = 1$



(a) Traffic ratios including service traffic

(b) Traffic ratios of token-based accounting

Figure 5.10: System-wide traffic ratios for $t = 17$, $k = 10$, $d_{var} = 1$

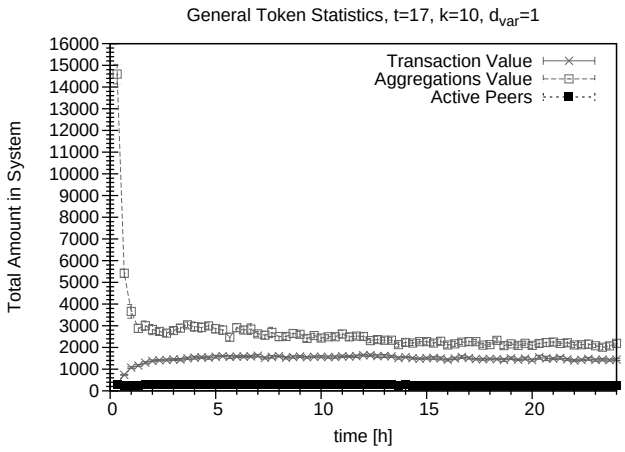
imately 9.96 MBytes per minute system-wide, which is per active peer approximately 22.77 kBytes per minute.

Maintenance Traffic Observations

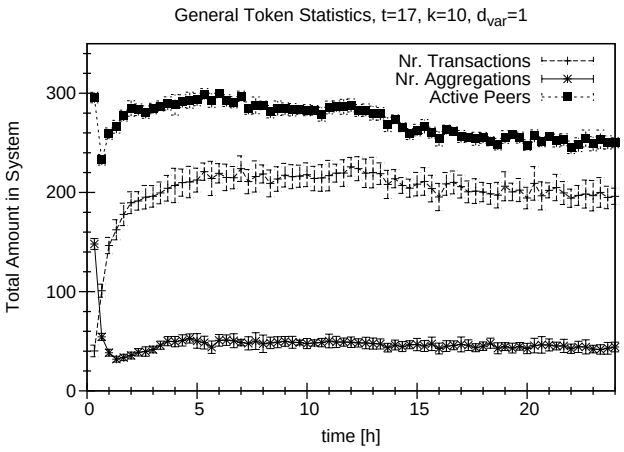
The breakdown of the account holder set maintenance traffic into the different mechanisms can be observed in Figure 5.12.

The highest amount of traffic stems from the account holder set check size mechanism. It increases with the number of aggregation accounts existing. System-wide traffic levels at approximately 1.16 MBytes per minute on average. At the end of the simulation this is approximately 2.87 kBytes traffic per minute per active peer on average.

Account movement creates the second highest amount of traffic among maintenance functions. Similar to the check size mechanism, account movement traffic increases with the number of existing aggregation accounts. System-wide account movement traffic levels at approximately 0.80 MBytes per minute on average. Per peer traffic is 1.98 kBytes per minute on average at the end of the simulation.



(a) Transactions, aggreagtions, active peers



(b) Transactions, aggreagtions, active peers (focused)

Figure 5.11: Token statistics, transactions, token aggregations for $t = 17$, $k = 10$, $d_{var} = 1$

The account locking and account consistency mechanisms create the third highest amount of traffic within account holder set maintenance. This traffic is closely related, because the mechanisms are executed consecutively. Due to the strong churn at the beginning of the simulation there are many changes in the Chord ring. Accordingly, the account holder sets hosting the aggregation accounts have to be adapted more often at the beginning of the simulation than afterwards. Account locking creates an average maximum system-wide traffic of 0.60 MBytes per minute; account consistency creates 0.54 MBytes. This is per peer, an average account locking traffic of 3.20 kBytes per minute and for account consistency 2.89 kBytes per minute. After the start-up phase, system-wide account locking traffic is 0.40 MBytes per minute on average; system-wide consistency traffic is 0.41 MBytes per minute on average. This results for account locking in a per peer traffic of 0.95 MBytes per minute on average, and 0.97 MBytes per minute on average for account consistency.

Traffic for checking the account positions is almost flat over the simulation time, only in the beginning it is slightly increased. Here, here are two effects that almost neutralise each other. In the beginning, there are fewer accounts that require position check, however this check is required more often due to system start-up and the related strong number of joins. Checking the aggregation accounts positions creates at a maximum, 0.23 MBytes system-wide traffic per minute on average. The maximum of average per peer traffic per minute is 0.79 kBytes.

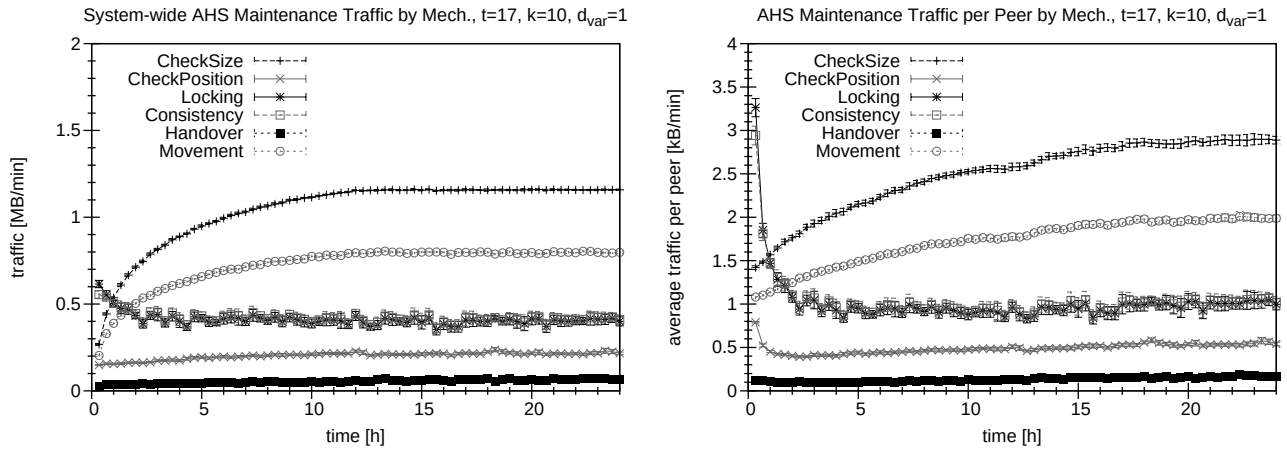
Account handovers create the least traffic with a average maximum of 0.07 MBytes system-wide per minute and 0.18 kBytes per peer per minute.

When the members of an account holder set change, update messages are sent to the old and new account holders. At the end of the simulation, the average update traffic per minute is 0.74 MBytes and per peer 2.87 kBytes per minute.

Observation of Single Peers

In order to observe the development of the peers' upload and download queue, the following results for the simulation runs that did not show any malfunction in Chord are presented, because these malfunctions would distort the results. In order to observe the traffic load that the token-based accounting scheme imposes on peers, for each peer the average queue length per minute was logged in the simulations without considering service traffic.[1]

[1] Averaging the results across the peers does not deliver more insights than observing the averaged traffic over the system or over several simulation runs, as a peer's online and offline time differs over the simulation runs.



(a) Maintenance traffic breakdown

(b) Maintenance traffic breakdown per peer

Figure 5.12: Maintenance traffic breakdown for $t = 17$, $k = 10$, $d_{var} = 1$

Figure 5.13 presents the proportions of the peers' average upload queue length per minute in seconds over the simulation runs that did not show the malfunction of Chord. Peers' upload bandwidth was set to 128 kBit per second. The results show that 97.85% of time, a peer has an upload queue length less 10 msec. In 1.21% of the time, the peers' upload queue length is smaller than 50 msec, and equal or larger than 10 msec. In 99.35% of time the upload queue length is below 1 second. Accordingly, peers have a very even traffic distribution.

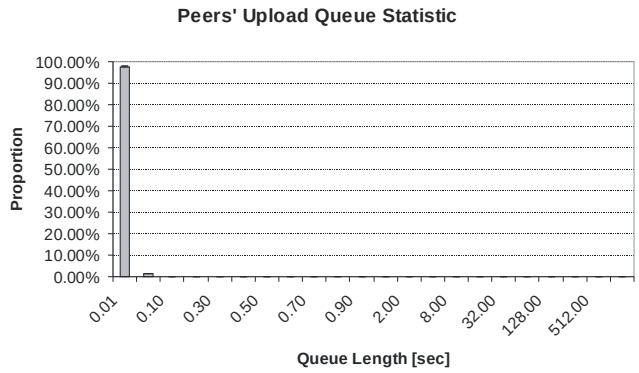
The times when a peer had an upload queue longer than one second for all simulation runs (excluding the simulation runs with malfunction of Chord) is shown in Figure 5.14. The very high peer loads happen at the beginning of the simulation, when many peers join, and their aggregation accounts and starting tokens are created. In a real life system such a strong join of new peers is highly unlikely; a system would grow slower. Accordingly, these high loads above 40 seconds upload queue lengths do not need to be considered further.

This analysis also shows that there are many queue lengths for peers in the range from one to approximately 15 seconds. All these queue lengths (but two) happened for trusted peers. Accordingly, the selection criteria for trusted peers should also take their capabilities into account, especially in terms of upload bandwidth. Still, a peers trustworthiness remains the primary selection criteria. Also, the impression that the graphs give might be misleading. The plotted queue length represents only 0.65% of all observations.

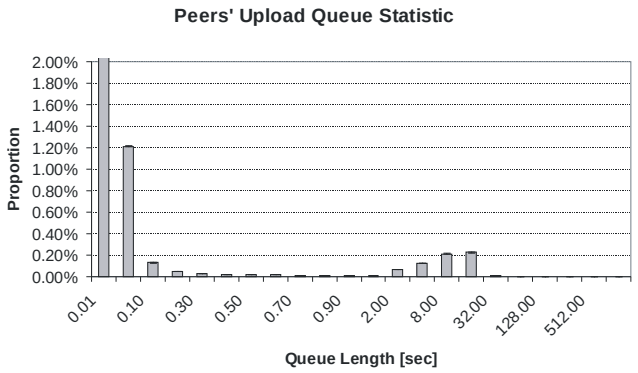
Finally, the peers' download queue lengths were observed in the same way as the upload queue length. A download bandwidth of 1024 kBit/sec was assumed for all peers. 99.43% of the time peers have a download queue length less than 10 msec. 99.96% of the time the download queue lengths were below 100 msec. The longest download queue length observed was between 0.7 seconds and 0.8 seconds. Graphs for these results have been omitted, as these do not add further insights to the pure numbers given.

Summary

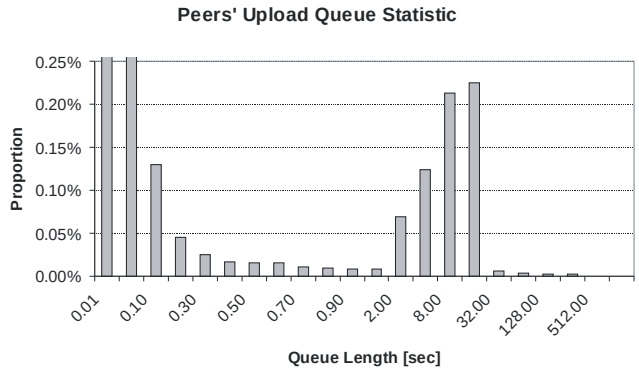
The analysis of the base experiment shows that the average traffic created by the token-based accounting scheme system-wide per transaction is approximately 293.58 kBytes. This traffic is distributed over several peers, as on average per transaction (apart from payment) there is an aggregation process and account holder set update to be considered. In the base experiment, there are at least 20 peers involved in all these processes, the 2 transaction partners, the token swapping peer, the aggregation administrator, the quorum, and one account holder set. Accordingly, the traffic is distributed widely among the



(a) Proportions of upload queue lengths, unfocused

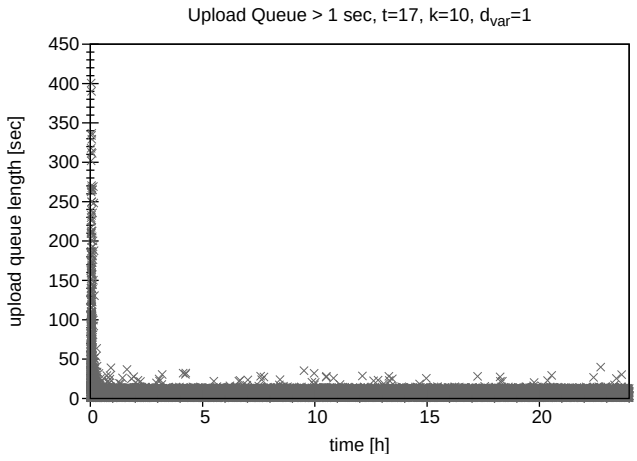


(b) Proportions of upload queue lengths, focused to lower 2%

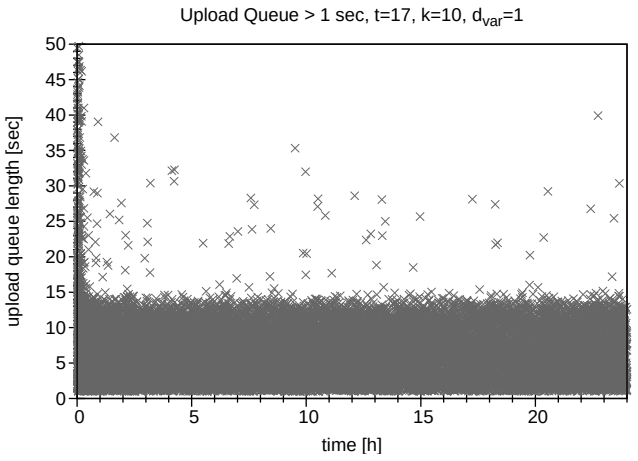


(c) Proportions of upload queue lengths, focused to lower 0.25%

Figure 5.13: Upload queue lengths percentage for the base experiment

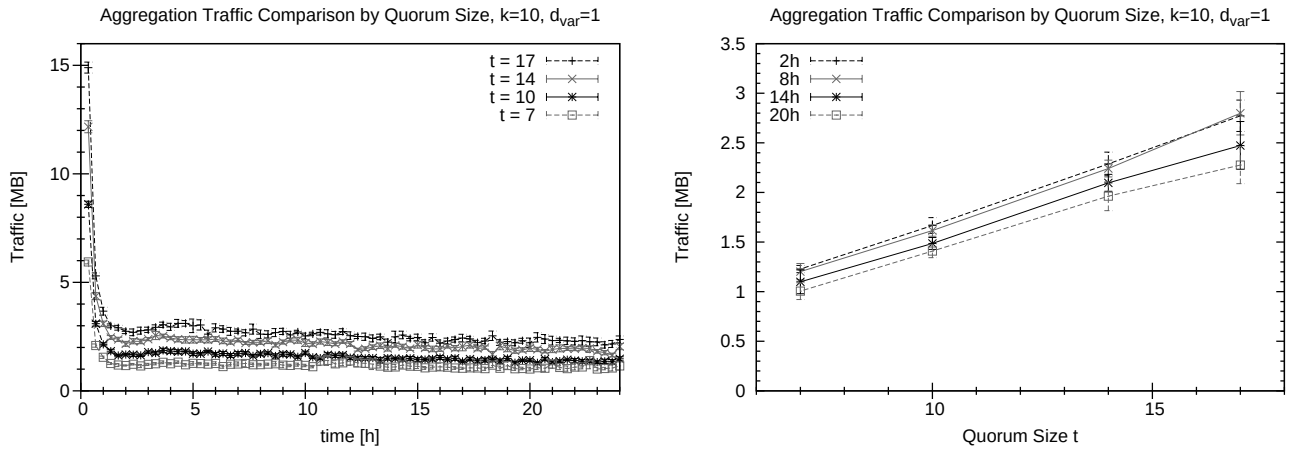


(a) Upload queue length by time



(b) Upload queue length by time, focused

Figure 5.14: High peer loads by simulation time



(a) Aggregation traffic by quorum size

(b) Aggregation traffic development

Figure 5.15: System-wide aggregation traffic by quorum size

participating peers. This also explains the low average queue length observed at the peers. Constant maintenance traffic per peer is approximately 22.77 kBytes per minute.

Furthermore, the peers' traffic load is very evenly distributed. Only trusted peers have to bear from time to time a load higher than their upload speed, however this happens only 0.64% of the time.

Overall, it can be noted, that the token-based accounting scheme can be used in this scenario to account for even services with low resource requirements because the induced traffic overhead per peer is low. Additionally, the highest proportion of traffic within the token-based accounting scheme is induced by Chord maintenance. This could be reduced using a different overlay network with less maintenance overhead. In the base experiment, the account holder set size is set larger then required for this churn. Accordingly, for a scenario with churn typically observed in p2p systems, this account holder set maintenance traffic would be reduced.

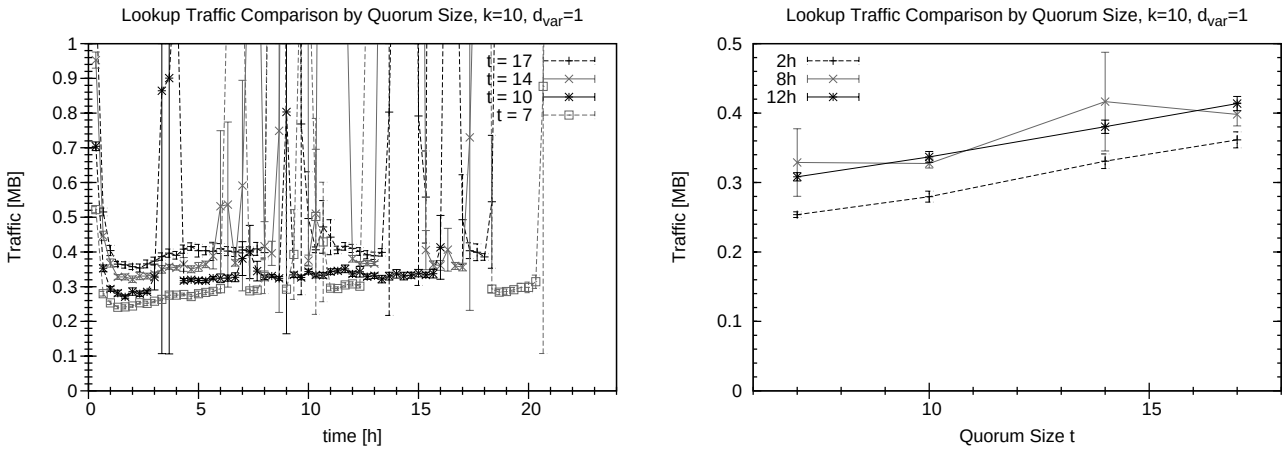
How the traffic overhead is develops with the changing scheme parameters will be discussed now.

5.4.3.3 Influence of the Quorum Size on Traffic Overhead

The quorum is only used with the token aggregation protocols. Accordingly, when the quorum size is altered, only the amount of token aggregation traffic and lookup traffic is changing, as token aggregation requires lookups.

Aggregation Traffic

Figure 5.15 shows the system-wide token aggregation traffic created on average per minute for the quorum sizes $t = \{17, 14, 10, 7\}$. The traffic development is approximately linear. With a quorum size of $t = 17$ the maximum system-wide aggregation traffic after the start-up phase is 3.06 MBytes per minute. With a quorum size of $t = 14$ the maximum system-wide aggregation traffic is approximately 2.52 MBytes per minute, with a quorum size of $t = 10$ the maximum system-wide aggregation traffic is approximately 1.86 MBytes per minute, and with a quorum size of $t = 7$ the maximum traffic is only 1.32 MBytes per minute. After the start-up phase the average token aggregation traffic per transaction for $t = 17$ results in 237.71 kBytes, for $t = 14$ in 196.64 kBytes, for $t = 10$ in 141.97 kBytes, and for t =7 in 111.46 kBytes. Analysing system-wide aggregation traffic results in a weak quadratic fit with $V_{aggr}(t) = 0.0011t^2 - 0.1281t + 0.2769$ MBytes with a coefficient of determination $R^2 = 1$.



(a) Lookup Traffic by Quorum Size

(b) Lookup traffic development

Figure 5.16: System-wide lookup traffic by quorum size

Lookup Traffic

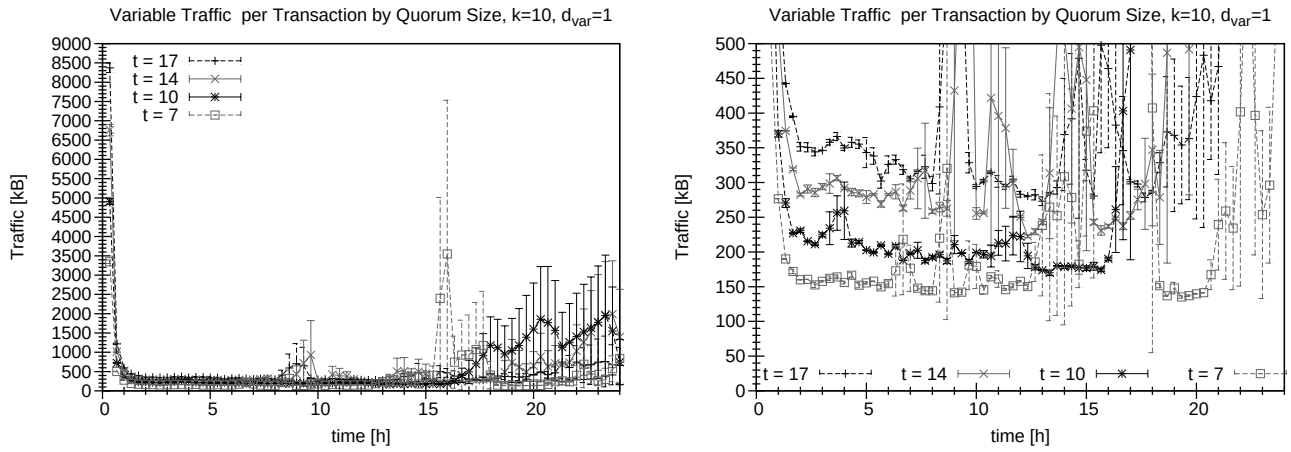
Analysing the development of the lookup traffic for the quorum size is complicated by the malfunction of Chord described above. Figure 5.16 a) shows the lookup traffic for the different quorum sizes. Figure 5.16 b) gives a clearer presentation for points in time when Chord is working correctly. Not considering the times where Chord malfunctions, with a quorum size of $t = 17$ the system-wide lookup traffic after the start-up phase is approximately 0.40 MBytes per minute. For lower quorum sizes lookup traffic decreases; this was expected. With a quorum size of $t = 14$ the system-wide lookup traffic is approximately 0.37 MBytes per minute, with a quorum size of $t = 10$ the system-wide lookup traffic is approximately 0.34 MBytes per minute, and with a quorum size of $t = 7$ the traffic is only 0.30 MBytes per minute. Lookup traffic per peer for $t = 7$ is 13.63 kBytes per minute on average, for $t = 10$ is 15.62 kBytes per minute, for $t = 14$ is 16.75 kBytes per minute, and for $t = 17$ is 17.89 kBytes per minute on average. Analysing system-wide lookup traffic results in a almost linear quadratic fit with $V_{lookup}(t) = 0.0001t^2 - 0.0079t + 0.2472$ MBytes, with a coefficient of determination $R^2 = 0.9999$.

Overall Traffic

The overall traffic splits into variable traffic per transaction and fixed traffic that happens due to system maintenance. Traffic per transaction is composed of payment traffic, aggregation traffic and the corresponding ratio of the lookup traffic. Fixed traffic is the remaining traffic.

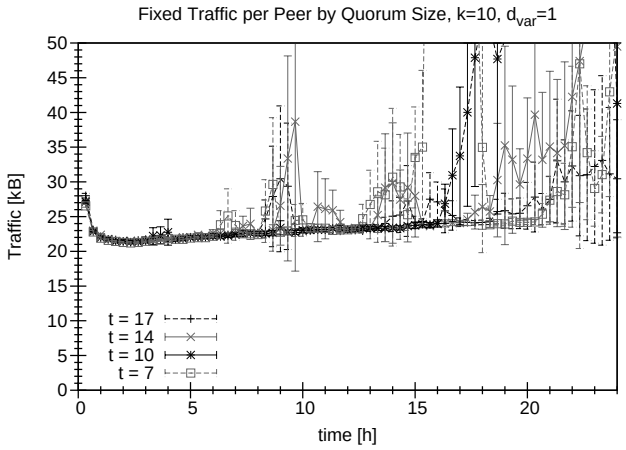
Figure 5.17 shows the comparison of the variable traffic per transaction by quorum size. Due to the malfunction of Chord this traffic varies strongly. Lookup traffic belonging to transactions is for $t = 17$ 56.37%, for $t = 14$ 52.23%, for $t = 10$ 44.77%, and for $t = 7$ 37.02% of all lookup traffic. At simulation time from 4 hours, 20 minutes to 6 hours, 20 minutes the variable traffic is on average 336.45 kBytes for $t = 17$, 281.60 kBytes for $t = 14$, 206.28 kBytes for $t = 10$, and 158.37 kBytes for $t = 7$. However, the variable traffic is still decreasing at that time. At a simulation time of 12 hours and 20 minute, where the variation is low, the variable traffic is is 280.45 kBytes for $t = 17$, 222.58 kBytes for $t = 14$, 194.37 kBytes for $t = 10$, and 150.05 kBytes for $t = 7$. This results in a best fit curve of $V_{trans}(t) = 0.3224t^2 + 4.478t + 107.04$ kBytess with a coefficient of determination $R^2 = 0.9723$; thus the development of variable traffic is linear. This is in accordance with the theoretical results from Section 5.3.

Figure 5.18 the average fixed traffic per minute per peer by quorum size. It does not vary by quorum size, because the quorum size does not influence the maintenance traffic, as mentioned above. This is in accordance with the theoretical results from Section 5.3. At simulation time from 4 hours, 20 minutes



(a) Variable traffic per transaction by quorum size (b) Variable traffic per transaction by quorum size (focused)

Figure 5.17: Variable traffic by quorum size



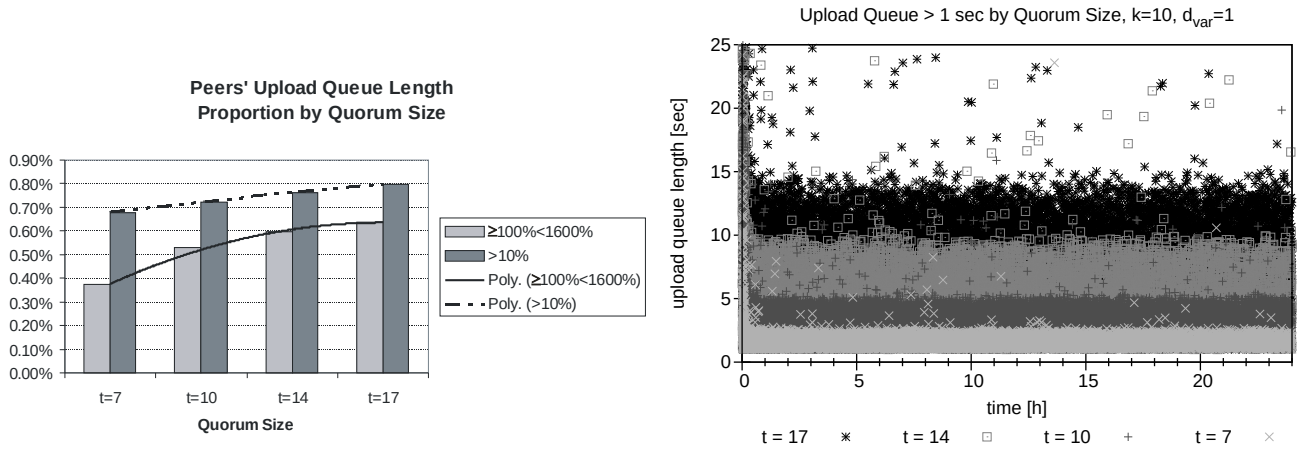
(a) Fixed traffic per peer by quorum size

Figure 5.18: Fixed traffic by quorum size

to 6 hours, 20 minutes fixed traffic is approximately 22.13 kBytes per minute per peer. Later in the simulation at 12 hours, 20 minutes it is approximately 23.21 kBytes per minute per peer.

Peer Load

The analysis of the peers' upload queues for the base experiment showed that trusted peers might experience a short congestion with up to 15 seconds upload queue length. Figure 5.19 shows the relevant statistics for the peers' upload queue length by quorum size. Figure 5.19 a) shows the proportions of upload queue length equal to or longer than 100 msec and from 1 second to 16 seconds. The increase of this proportion gets smaller with larger quorum size. For $t = 7$ 99.32% of time peers have an average upload queue less than 100 msec per minute. For $t = 10$ this is 99.28%, for $t = 14$ this is 99.24%, and for $t = 17$ this is 99.20%. The best fit curve for to this is $p_{UpQL<100msec} = 0.000002t^2 - 0.0002t + 0.9942\%$ with a coefficient of determination $R^2 = 0.9947$. Accordingly, the proportion of low upload queue length decreases very little, almost linearly with increased quorum size. Analysing the peers' queue lengths from one second to 16 seconds shows that for $t = 7$ this queue length happens at 0.37% of time, for $t = 10$ at



(a) Proportion of queue length by quorum size

(b) Queue lengths > 1 ms by simulation time by quorum size

Figure 5.19: Peers' upload queue length proportions by quorum size, $k = 10$

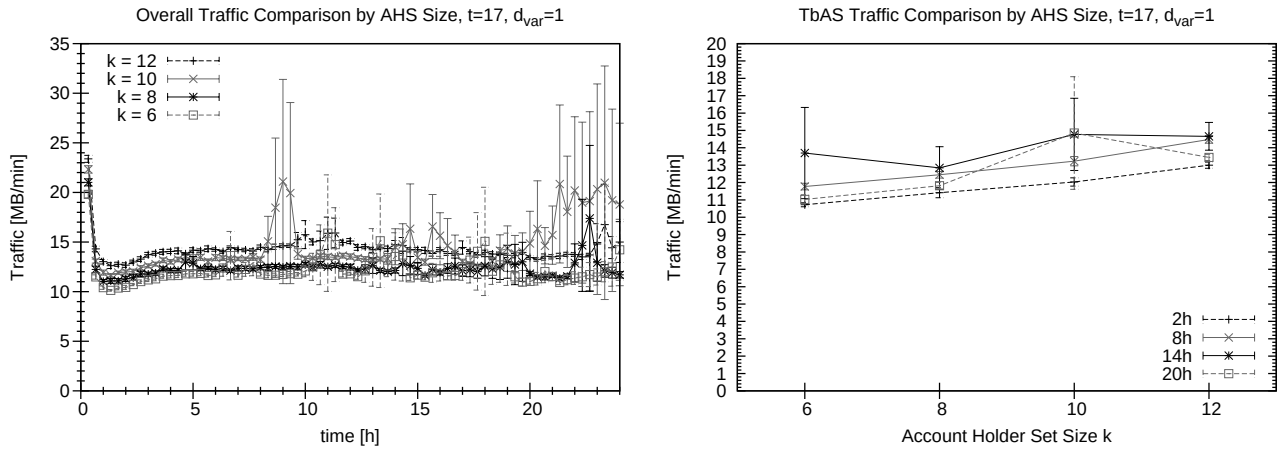
0.53% of time, for $t = 14$ at 0.60% of time , and for $t = 17$ at 0.64% of time. The best fit curve for this is $p_{1sec \leq UpQL < 16sec} = -0.00003t^2 + 0.0009t - 0.0012\%$ with a coefficient of determination $R^2 = 0.9875$. The proportion of this upload queue length range increases less with increasing quorum size. However, Figure 5.19 b) shows the distribution of upload queue length within this range. A border can be observed in the graphs where above it, upload queue length becomes distinctively more unlikely. For $t = 7$ this boarder is at approximately 3 seconds of upload queue length, for $t = 10$ the border is at approximately 5 seconds of upload queue length, for $t = 14$ the border is at approximately 9.5 seconds of upload queue length, and for $t = 17$ the border is at approximately 14 seconds of upload queue length. The best fit curve for this is $p_{100\% \leq UpQL < 1600\%} = 0.0006t^2 - 0.0033t + 0.0234$ sec with a coefficient of determination $R^2 = 0.9999$. Accordingly, the border upload queue length increases with increasing quorum size.

Summary

The experiments varying the quorum size shows the influence of the desired Trust Level of the token-based accounting scheme on the traffic induced by it. The simulation results support the results from the analytic evaluation. Apart from token aggregation and lookups, traffic remains constant. Both aggregation traffic and lookup traffic increases only slightly with a weak quadratic factor with increased quorum size. Overall, system-wide aggregation traffic with 237.71 kBytes of traffic per transaction for the maximum traffic scenario $t = 17$, $k = 10$ is a practical value, because this traffic is distributed among the peers participating in a token aggregation process. In this scenario there are 29 different peers involved (swapping peer, aggregation administrator, quorum peers, and account holders).

For an increased quorum size the load of the peers expressed by their upload queue length increases only slightly on average. However, trusted peers might experience maximum upload queue lengths of several seconds, with a low proportion of 0.64%. This upload queue length increases from 3 seconds for $t = 7$ to 14 seconds $t = 17$. Accordingly, for systems that require a very high Trust Level, and systems where the ratio of honest trusted peers is estimated to be low, it is important that the upload capability of peers is also taken into account, when trusted peers are selected.

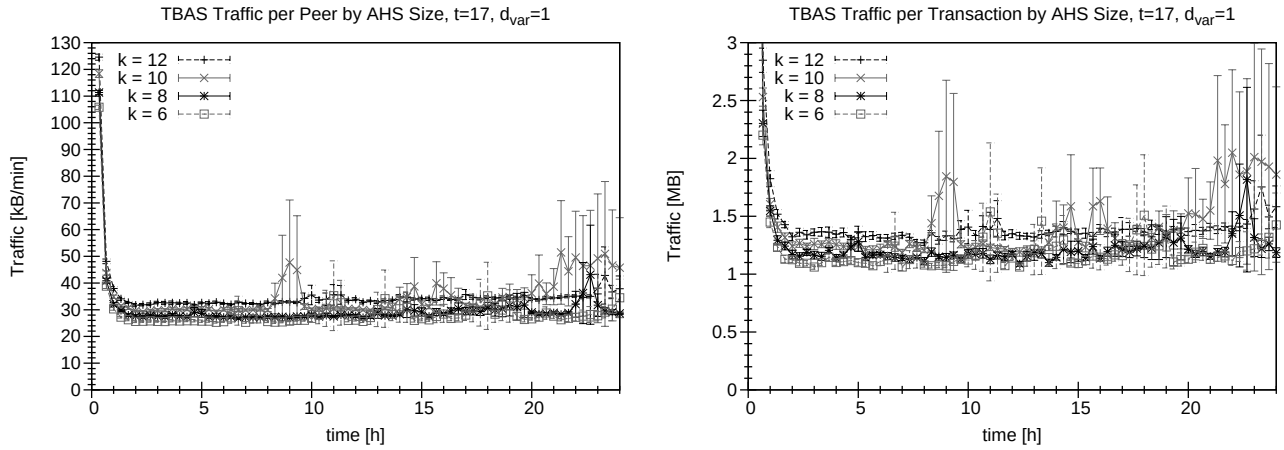
Accordingly, the token-based accounting scheme can be used accounting services that consume small amounts of a peer's resources.



(a) TBAS traffic by AHS size

(b) TBAS traffic development by AHS size

Figure 5.20: Total token-based accounting traffic by AHS size



(a) Total TBAS traffic per peer

(b) TBAS traffic by transaction

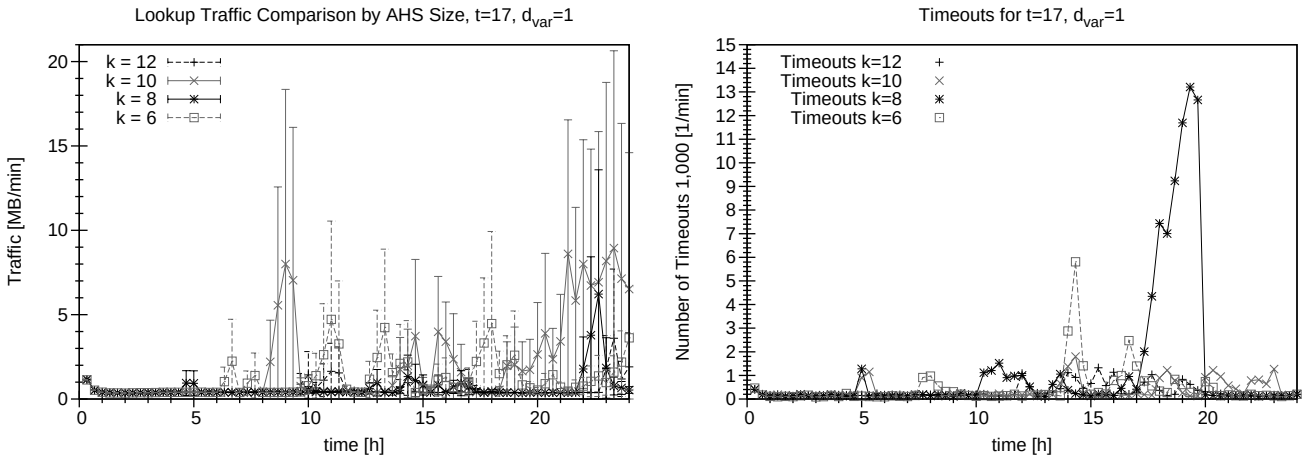
Figure 5.21: Total traffic per peer and per transaction by AHS size

5.4.3.4 Influence of the Account Holder Set Size on Traffic Overhead

The account holder sets which store the aggregation accounts are involved within the token-based accounting scheme in token payment and token aggregation and also have their own maintenance protocols. Account holder set maintenance creates within the token-based accounting scheme the largest proportion of traffic (see above). In the following, the traffic development for the different mechanisms is analysed.

Total Token-based Accounting Traffic

Figure 5.20 shows the total traffic created by the token-based accounting scheme for the account holder set sizes $k = \{12, 10, 8, 6\}$. With a growing account holder set, the traffic grows slowly exponentially. This was expected according to the analytical traffic assessment of the account holder set maintenance mechanisms, because some of them have a traffic complexity of $O(k^2)$. In detail, on average over the simulation time, per minute the system-wide 14.22 MBytes traffic is created for an account holder set size of $k = 12$, 13.39 MBytes for a $k = 10$, 12.23 MBytes for a $k = 8$, and 11.67 MBytes for a



(a) Lookup traffic by AHS size

(b) Timeouts by AHS experiments

Figure 5.22: Reasons for volatile aggregation traffic for AHS size $k = 6$

$k = 6$. The average traffic per minute per peer for $k = 12$ is 32.37 kBytes, for $k = 10$ is 30.10 kBytes, for $k = 8$ is 27.22 kBytes, and for $k = 6$ is 25.55 kBytes (see Figure 5.21 a)). Thus, although the account holder set size is doubled, the traffic does not double. A functional fit to the traffic development in Figure 5.20 b) results in $V(k) = 0.0167k^2 + 0.1404k + 10.182$ MBytes with a coefficient of determination $R^2 = 0.9893$.

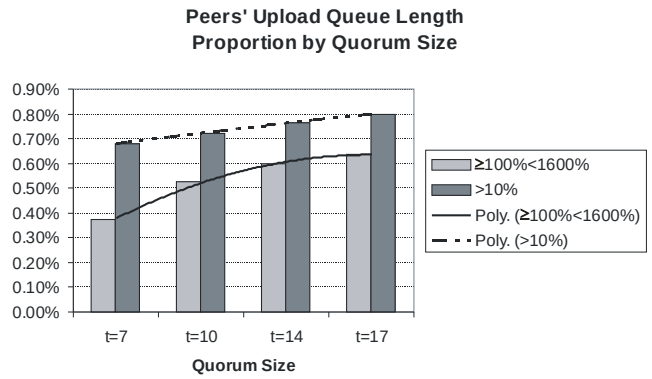
The outliers which can be observed in Figure 5.20 result from the described malfunction of the Chord overlay, as can be observed in Figure 5.22 a).

Figure 5.21 b) shows the traffic generated by transactions. The average amount of traffic generated per transaction is approximately 1.37 MBytes for $k = 12$, 1.27 MBytes for $k = 10$, 1.20 MBytes for $k = 8$, and 1.17 MBytes for $k = 6$. This, per peer, is 3.27 MBytes for $k = 12$, 2.97 kBytes for $k = 10$, 2.80 kBytes for $k = 8$, and 2.69 kBytes for $k = 6$. These figures also consider Chord maintenance traffic.

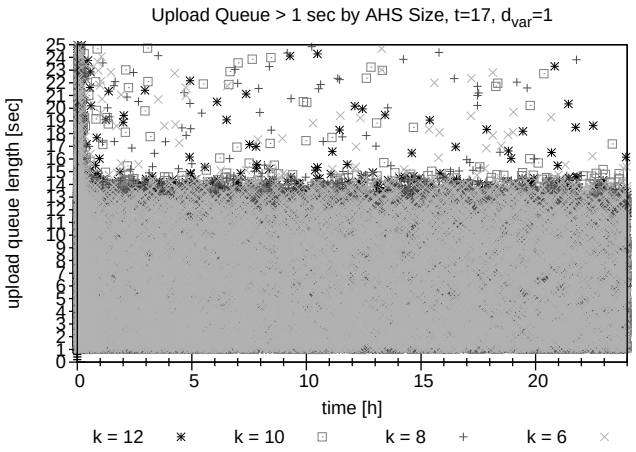
Analysing the peers' loads by observing their upload queue length shows that there is only a small influence of the account holder set size. Figure 5.23 a) shows the proportions of upload queue length equal to or longer than 100 msec and from 1 second to 16 seconds. The increase from account holder set size 6 to account holder set size 12 decreases with larger account holder set size. For $k = 6$ 99.25% of time peers have an average upload queue less than 100 msec per minute. For $k = 8$, this is 99.20%, for $k = 10$, this is also 99.20%, and for $k = 12$, this is 99.17%. The best fit curve for this is $p_{UpQL<100msec} = 0.000008k^2 - 0.0003k + 0.9937\%$ with a coefficient of determination $R^2 = 0.8991$. Accordingly, the proportion of low upload queue length decreases very little, almost linearly with increased quorum size. Analysing the peers' queue lengths from one second to 16 seconds shows that for $k = 6$ this queue length happens at 0.61% of time, for $k = 8$ at 0.62% of time, for $k = 10$ at 0.63% of time , and for $k = 12$ at 0.62% of time. The best fit curve for this is $p_{1sec \leq UpQL<16sec} = -0.00005k^2 + 0.0002x + 0.0052\%$ with a coefficient of determination $R^2 = 0.7617$. The proportion of this upload queue length range increases less with increasing account holder set size and cannot be recognised between $k = 10$ and $k = 12$. Figure 5.23 b) shows the distribution of upload queue length within this range, which does not change in a significant way over the account holder set sizes. Accordingly, the account holder set size has only a very small influence on the peer's load, which is insignificant compared to the influence of the quorum size.

Payment

As mentioned, token payment traffic is influenced only marginally by the account holder set size, as Figure 5.24 confirms. For all account holder set sizes the average amount of traffic created per

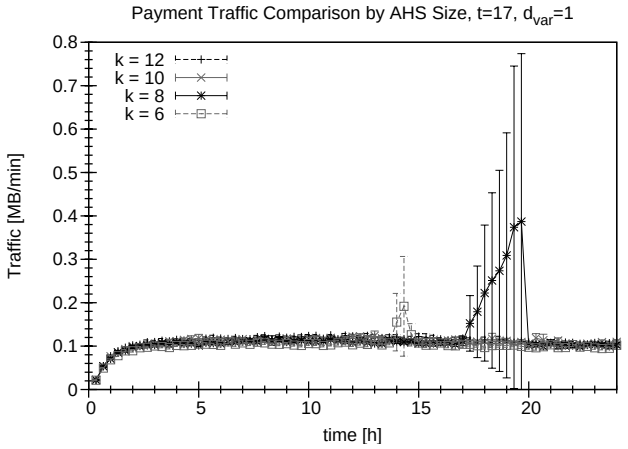


(a) Proportion of queue length by AHS size

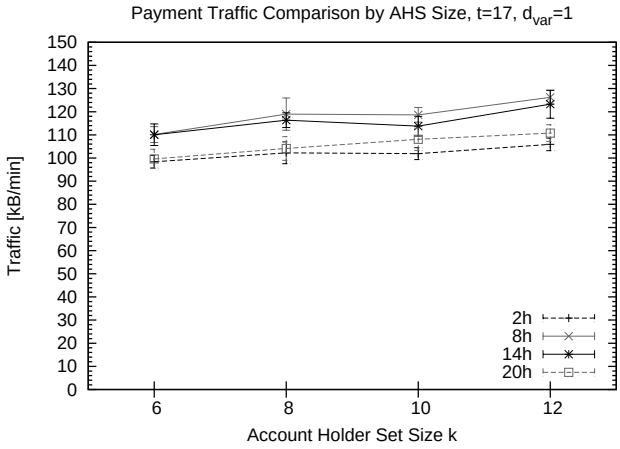


(b) Queue lengths > 1 ms by simulation time by AHS size

Figure 5.23: Peer's upload queue length proportions by AHS size, $t = 17$



(a) Payment traffic by AHS size



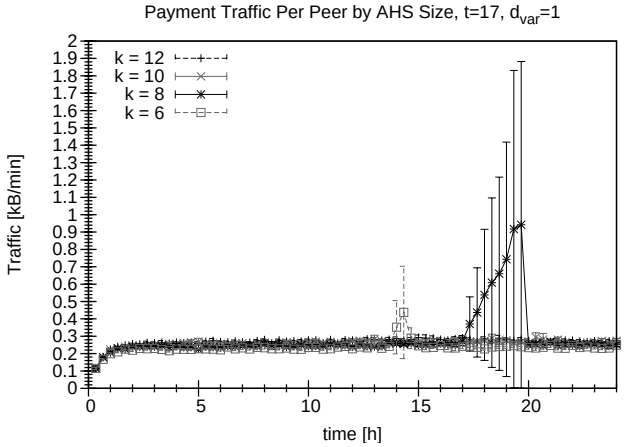
(b) Payment traffic development by AHS size

Figure 5.24: Payment traffic by AHS size

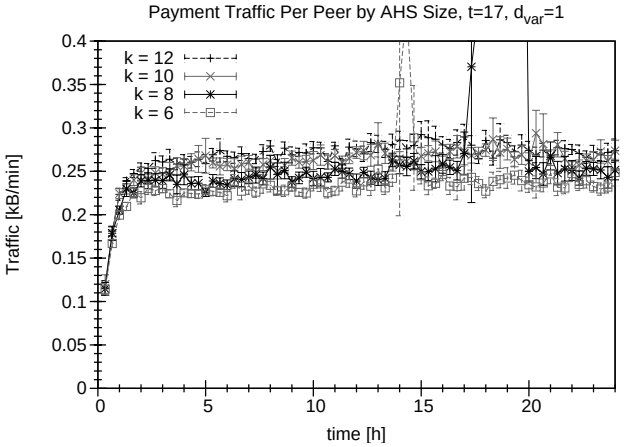
transaction is between 0.106 and 0.115 MBytes per transaction during the simulation time. The best fit curve for the traffic development in Figure 5.24 b) is $V(k) = 0.00005k^2 - 0.0009k + 0.0983$ MBytes with a coefficient of determination $R^2 = 0.9576$.

The increased payment traffic for account holder set size $k = 8$ between 17 and 20 hours of simulation time and the increased payment for account holder set size $k = 6$ between 13 and 15 hours of simulation time is due to timeouts during the payment process. Figure 5.22 shows this. The timeouts lead to a repeated sending of messages. The strong increase in traffic is due to an unresolved bug in the simulation where nested timeouts call methods that re-send payment messages.

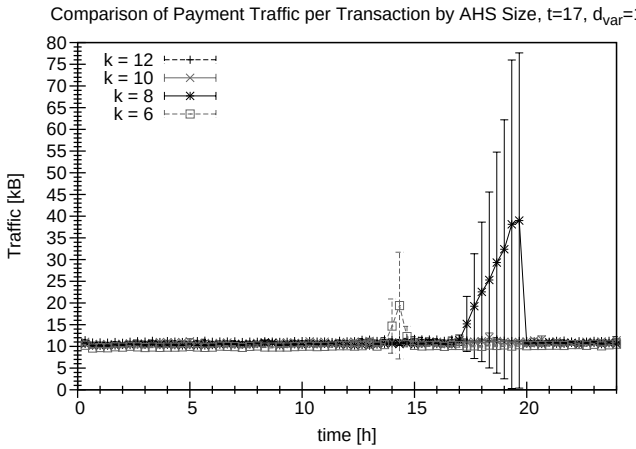
Figure 5.25 shows the average payment traffic generated per minute, per peer, and per transaction. Not considering the traffic due to the timeouts, the average payment per peer per minute is between 0.24 kBytes for $k = 6$ and 0.27 kBytes for $k = 12$. Payment traffic per transaction is depicted in Figure 5.25 c) and d). Per transaction 10.13 kBytes of payment traffic is generated for $k = 6$ and 11.28 kBytes for $k = 12$.



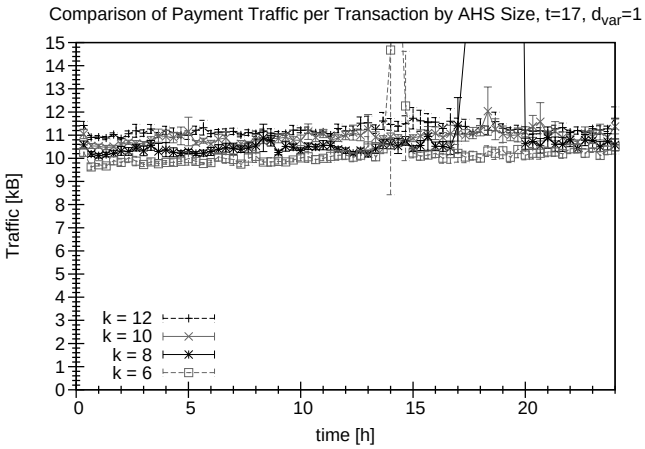
(a) Payment traffic per peer by AHS size



(b) Payment traffic per peer by AHS size, focused

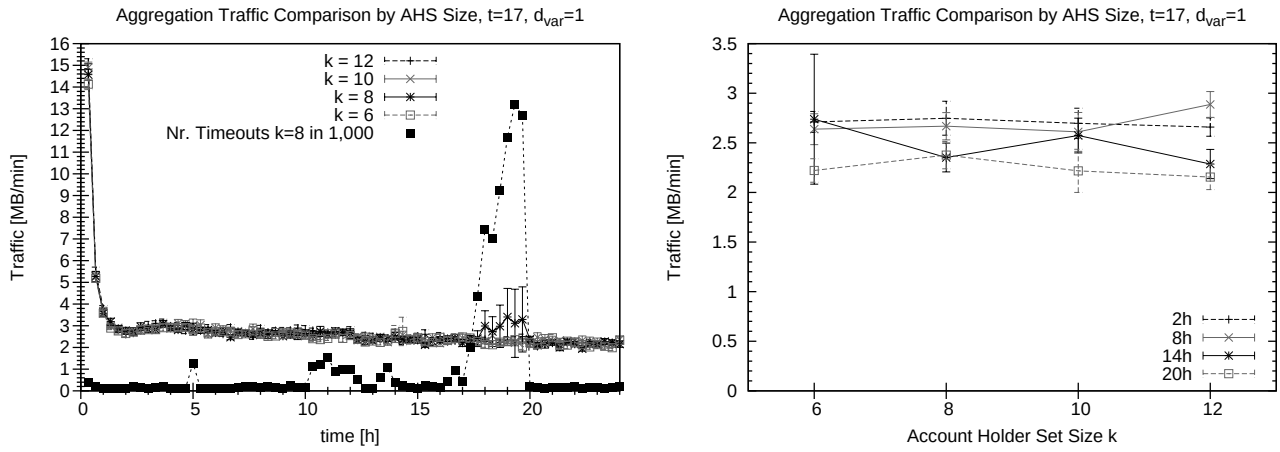


(c) Payment traffic per transaction by AHS size



(d) Payment traffic per transaction by AHS size, focused

Figure 5.25: Payment traffic per peer and per transaction



(a) Token aggregation traffic by AHS size (b) Aggregation traffic development by AHS size

Figure 5.26: Token aggregation traffic by AHS size

Token Aggregation

Figure 5.26 shows the aggregation traffic created by account holder set size. The volatile results of the simulations for $k = 8$, which result in an increased traffic, have been explained above. The average system-wide aggregation traffic per minute between 3 hours and 6 hours simulation time is 2.85 MBytes for $k = 6$, 2.88 MBytes for $k = 8$, 2.93 MBytes for $k = 10$, and 2.94 MBytes for $k = 12$. During this simulation time the average aggregation traffic per minute per peer for $k = 6$ is 3.36 kBytes, for $k = 8$ is 6.55 kBytes, for $k = 10$ is 6.75 kBytes, and for $k = 12$ is 6.84 kBytes.

During this simulation time the average aggregation traffic per transaction is 277.44 kBytes for $k = 6$, 284.21 kBytes for $k = 8$, 285.55 kBytes for $k = 10$, and 289.32 kBytes for $k = 12$.

All Account Holder Set Maintenance Mechanisms

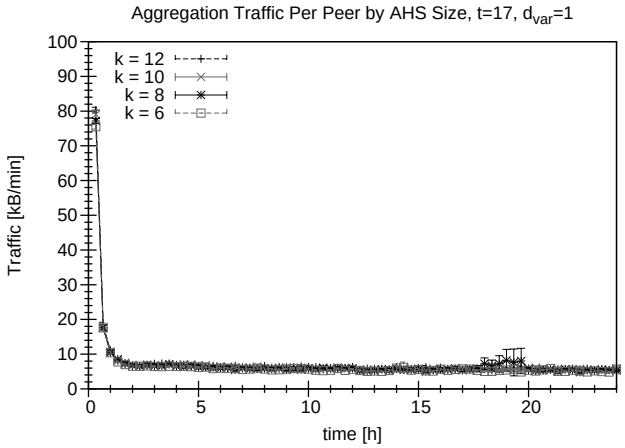
The account holder sets are maintained using a collection of mechanisms. Accordingly, the account holder set size has direct influence on the traffic generated by these mechanisms. Figure 5.26 shows how the traffic develops for the set sizes $k = \{6, 8, 10, 12\}$. The maintenance traffic increases over the simulation time due to the increase in number of aggregation accounts. The maintenance traffic flattens after approximately 20 hours of simulation time. When the maintenance traffic flattens, all maintenance mechanisms create in total an average with $k = 6$ of 1.38 MBytes system-wide traffic per minute, with $k = 8$ of 2.15 MBytes system-wide traffic, with $k = 10$ of 3.08 MBytes system-wide traffic, and with $k = 12$ of 4.22 MBytes system-wide traffic. Overall, that is an increase of factor 3.06 for a doubled account holder set size. Figure 5.26 c) shows the traffic development more clearly. A fit to this results in $V_{AHS\,Maintenance} = 0.0225k^2 + 0.0674k + 0.1647$ MBytes with a coefficient of determination $R^2 = 1$. The maintenance traffic per peer is depicted in Figure 5.26 c). On average per minute per peer for $k = 6$ there are 3.35 kBytes, $k = 8$ there are 5.30 kBytes, $k = 10$ there are 7.70 kBytes, $k = 12$ there are 10.74 kBytes of maintenance traffic created.

Maintenance traffic is also effected by churn, which will be analysed later. It is important to note that for the base churn scenario, an account holder set size of $k = 6$ is sufficient.

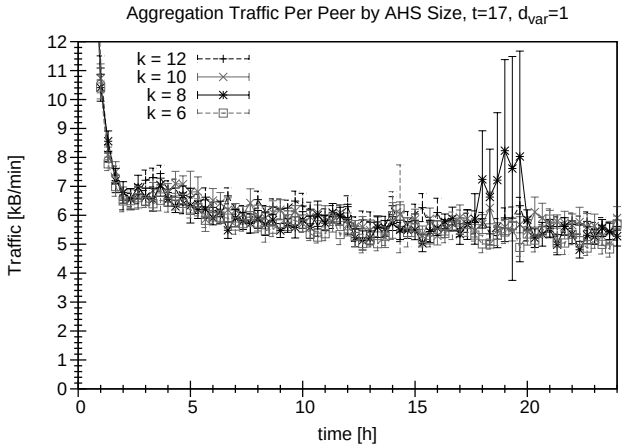
The different mechanisms of account holder set maintenance are analysed now.

Account Holder Set Check Size Mechanism

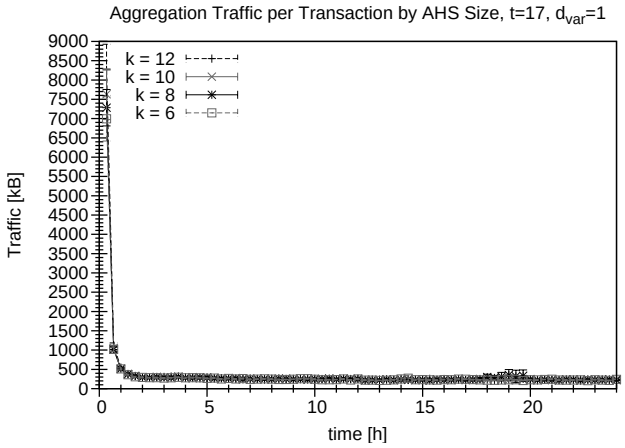
The check of the size of the account holder set size is executed periodically for each set. The traffic created by account holder set size is depicted in Figure 5.29. The flat check position traffic at the end of the simulation time is on average per minute for $k = 6$ 0.56 MBytes, for $k = 8$ it is 0.84 MBytes , for



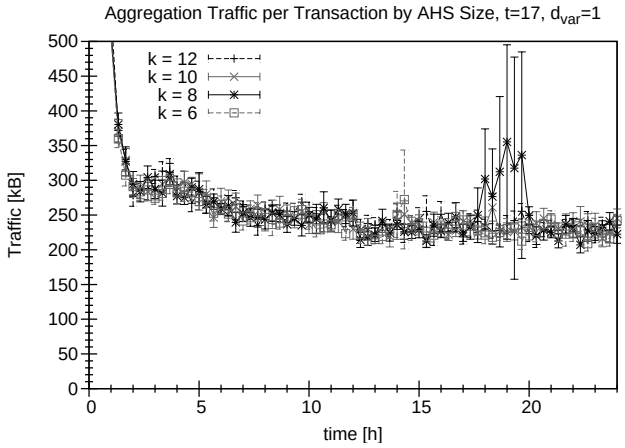
(a) Token aggregation traffic per peer by AHS size



(b) Token aggregation traffic per peer by AHS size, focused

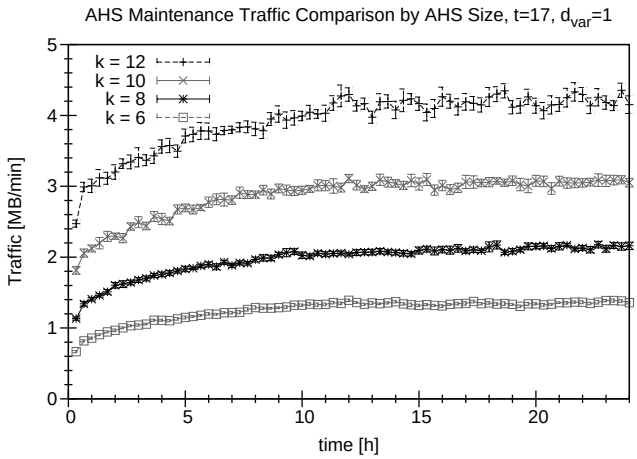


(c) Token aggregation traffic per transaction by AHS size

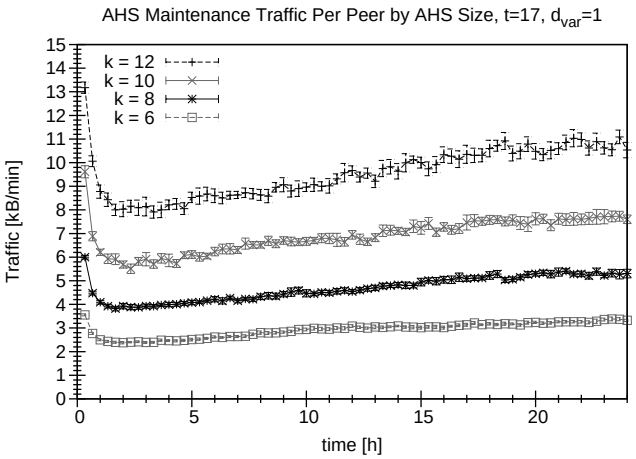


(d) Token aggregation traffic per transaction by AHS size, focused

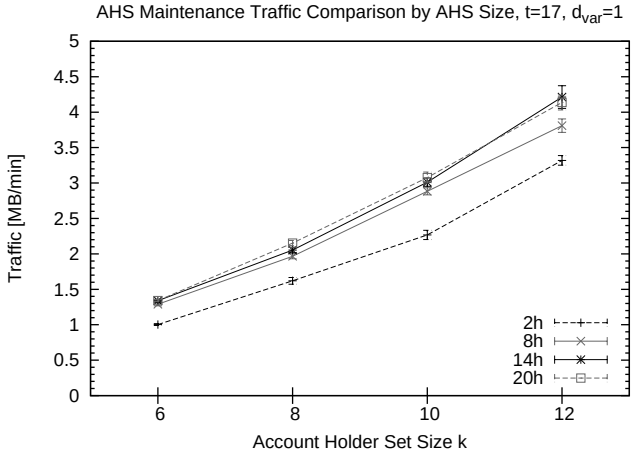
Figure 5.27: Token aggregation traffic per peer and per transaction by AHS Size



(a) AHS maintenance traffic by AHS size

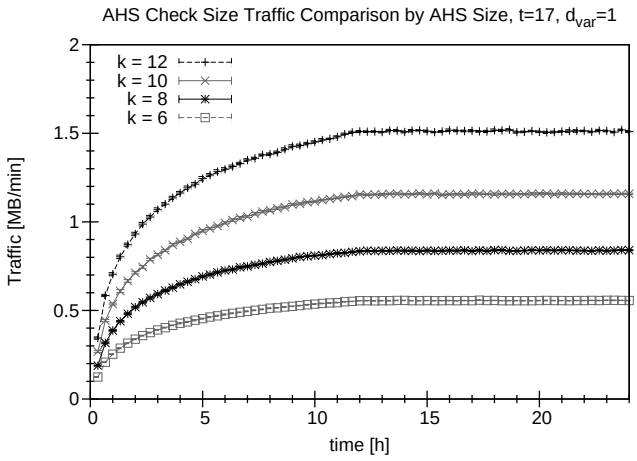


(b) AHS maintenance traffic per peer by AHS size

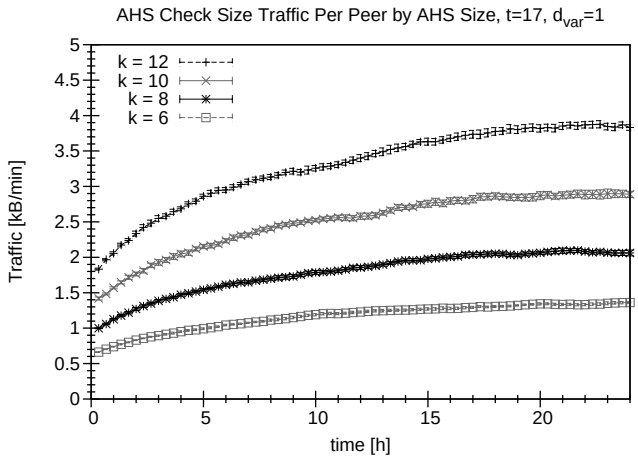


(c) Maintenance traffic development by AHS size

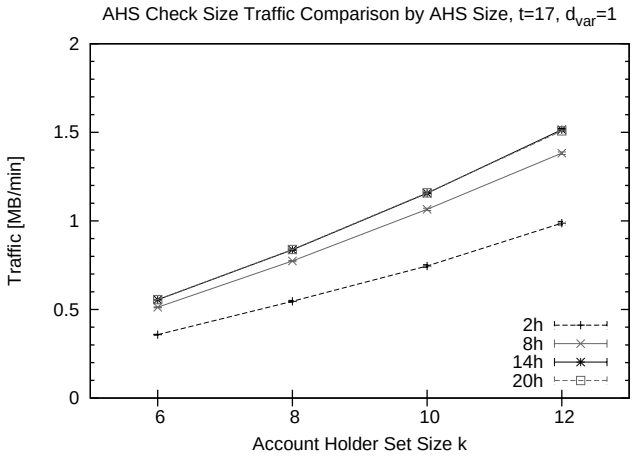
Figure 5.28: Account holder set maintenance traffic by AHS size



(a) Account holder set check size traffic by AHS size



(b) Account holder set check size traffic per peer by AHS size



(c) AHS check size traffic development by AHS size

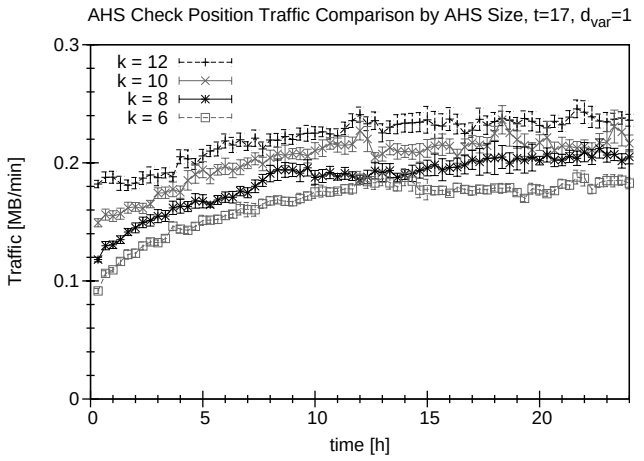
Figure 5.29: Account holder set check size traffic by AHS size

$k = 10$ it is 1.16 MBytes, and for $k = 12$ it 1.51 MBytes. The traffic increases by factor 2.7 for a doubled account holder set size. This development is depicted in Figure 5.30 c). The curve fit for this results to $V_{AHS\,CheckPosition} = 0.0045k^2 + 0.0779k - 0.074$ MBytes with a coefficient of determination $R^2 = 1$. This shows that there is quadratic factor in the curve, which can also be estimated from the figure. The average traffic per peer per minute is depicted in Figure 5.30 b). For $k = 6$ 1.35 kBytes of traffic are created, for $k = 8$ 2.07 kBytes of traffic are created, for $k = 10$ 2.89 kBytes of traffic are created, and for $k = 12$ 3.85 kBytes of traffic are created per peer.

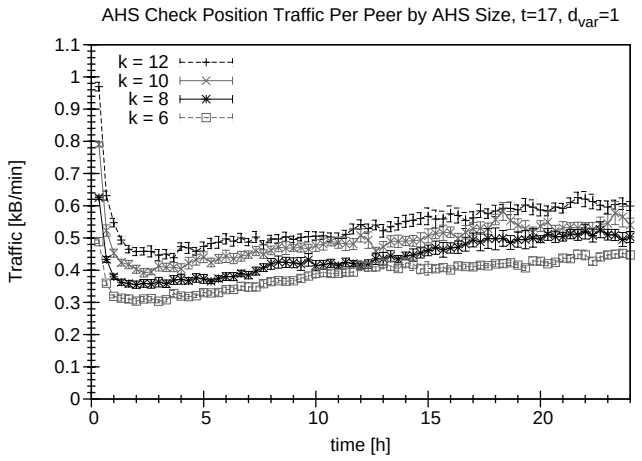
Account Holder Set Check Position Mechanism

Like the check size mechanism, the check position mechanism for account holder sets is executed periodically for each set. Similar to the check size traffic, the check position traffic increases over the simulation time due to the increase in the number of aggregation accounts and it flattens after approximately 20 hours of simulation time.

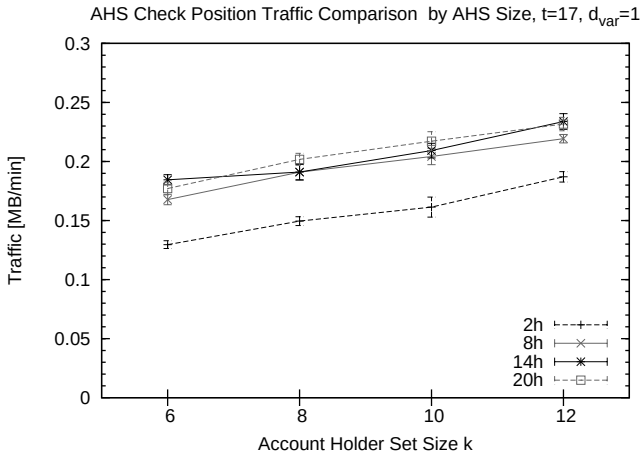
The system-wide average traffic created per minute is depicted in Figure 5.30 a). The flat check position traffic at the end of the simulation time on average per minute for $k = 6$ is 0.18 MBytes, for $k = 8$ it is 0.21 MBytes , for $k = 10$ it is 0.22 MBytes, and for $k = 12$ it is 0.24 MBytes. The traffic



(a) Account holder set check position traffic by AHS size



(b) Account holder set check position traffic per peer by AHS size



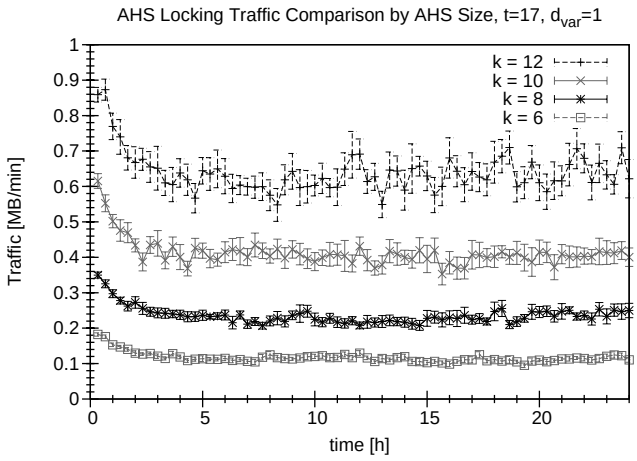
(c) AHS check position traffic development by AHS size

Figure 5.30: Account holder set check position traffic by AHS size

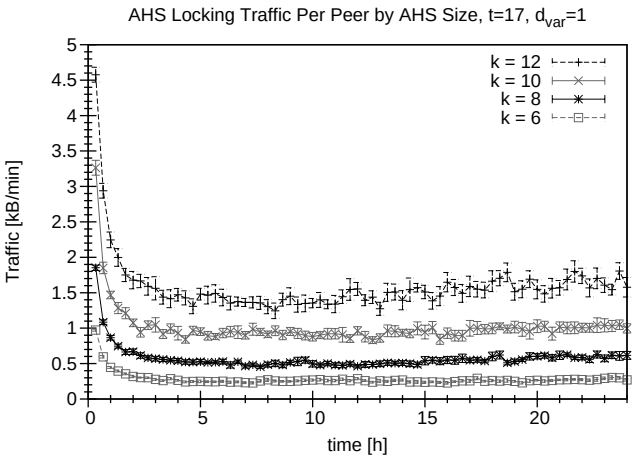
increases by a factor of 1.3 for a doubled account holder set size. This development is depicted in Figure 5.30 c). The best fit curve for this results in $V_{AHS\,CheckPosition} = -0.0005k^2 + 0.0172k + 0.0988$ MBytes with a coefficient of determination $R^2 = 0.9985$. The check size per peer is highest at the beginning of the simulation and approximately half that at the end of the simulation. At the end of the simulation, the check position traffic per peer on average for $k = 6$ is 0.45 kBytes per minute, for $k = 8$ is 0.51 kBytes per minute, for $k = 10$ is 0.55 kBytes, and for $k = 12$ is 0.60 kBytes.

Lock Aggregation Account Mechanism

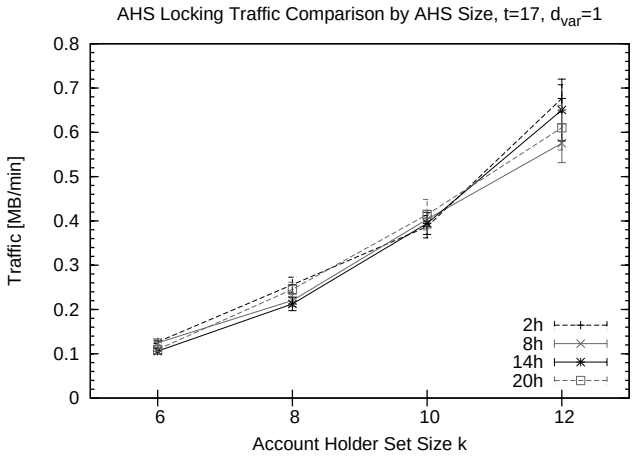
The mechanism locking an account holder set is only executed if required. Furthermore, the message complexity of this mechanism is $O(k^2)$. Therefore, the traffic for this mechanism can be more volatile than for other maintenance mechanisms. Figure 5.31 a) depicts the average system wide traffic created per minute by the aggregation account locking mechanism. The locking traffic is highest at the beginning of the simulation, because due to the high number of joins at the beginning of the simulation, a higher number of aggregation account repositions are required than during the remaining simulation time. At the end of the simulation on average with $k = 6$ there is 0.12 MBytes of system-wide traffic created per



(a) Lock aggregation account traffic by AHS size



(b) Lock aggregation account traffic per peer by AHS size



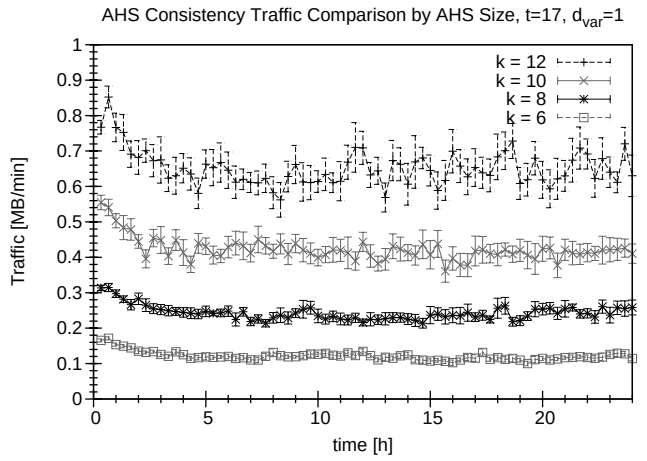
(c) Aggregation account locking traffic development by AHS size

Figure 5.31: Lock aggregation account traffic by AHS size

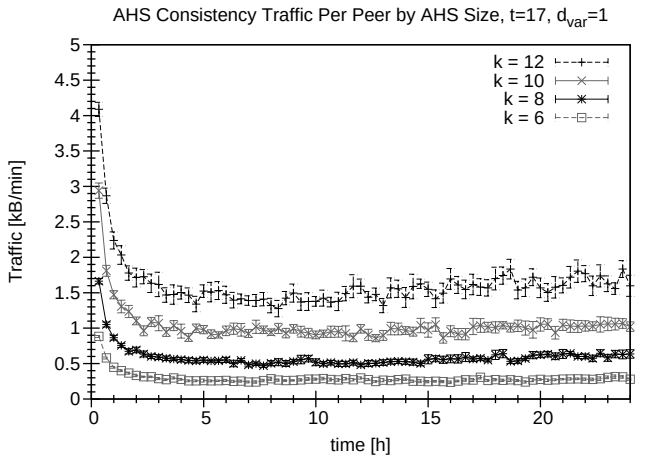
minute by aggregation account locking, with $k = 8$ there are 0.25 MBytes created, with $k = 10$ there are 0.41 MBytes created, and with $k = 12$ there are 0.65 MBytes created. Accordingly, traffic increases by a factor of 5.4 for a double account holder set size. Figure 5.31 b) shows this development in more detail. The best fit curve results in $V_{AHS\,Locking} = 0.0066k^2 - 0.0319k + 0.0724$ MBytes with a coefficient of determination of $R^2 = 0.9997$. Also, the lock aggregation account traffic per peer is highest at the beginning of the simulation and approximately one third of that at the end of the simulation. At the end of the simulation the account locking traffic per peer for $k = 6$ is 0.29 kBytes per minute, for $k = 8$ is 0.61 kBytes per minute, for $k = 10$ is 1.03 kBytes per minute, and for $k = 12$ is 1.65 kBytes per minute.

Aggregation Account Consistency Mechanism

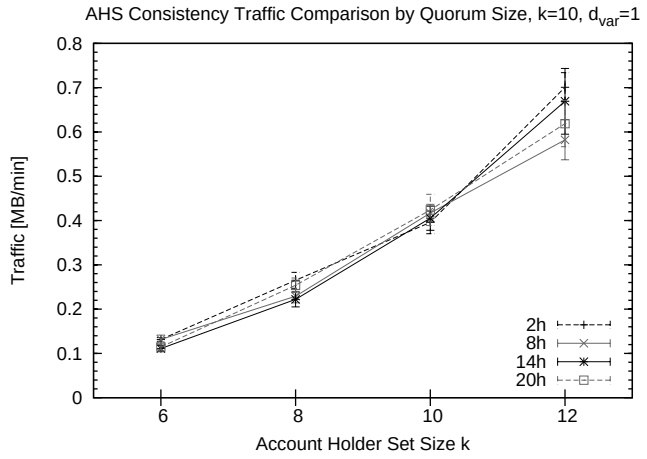
Similar to account locking, the account consistency mechanism has a message complexity of $O(k^2)$. This mechanism is only executed on demand and in sequence with account locking. Therefore, its traffic is highest at the beginning at the simulation. The traffic generated by this mechanism is depicted in Figure 5.32 a). At the end of the simulation the average system-wide traffic for $k = 6$ is 0.12 MBytes per minute, for $k = 8$ is 0.25 MBytes per Minute, for $k = 10$ is 0.42 MBytes per minute, and for $k = 12$



(a) Aggregation account consistency traffic by AHS size



(b) Aggregation account consistency traffic per peer by AHS size



(c) Aggregation account consistency traffic development by AHS size

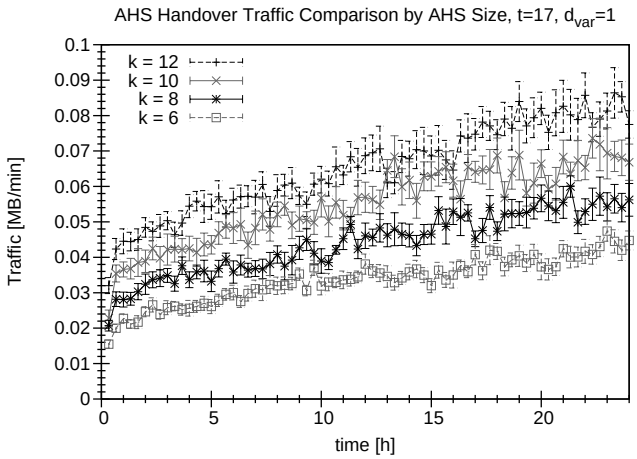
Figure 5.32: Aggregation account consistency traffic by AHS size

is 0.66 MBytes per minute. Overall, aggregation account locking traffic is increased by a factor of 5.4 for a doubled account holder set size. This development is depicted in more detail in Figure 5.32 b). The best fit curve for this is $V_{AHS\,Locking} = 0.0066k^2 - 0.0303k + 0.0693$ MBytes with a coefficient of determination $R^2 = 0.9996$. Also the account consistency traffic per peer is highest at the beginning of the simulation and approximately more than double as high as at the end of the simulation. At the end of the simulation the account consistency traffic per peer for $k = 6$ is 0.30 kBytes per minute, for $k = 8$ is 0.25 kBytes per minute, for $k = 10$ is 0.42 kBytes per minute, and for $k = 12$ is 0.66 kBytes per minute.

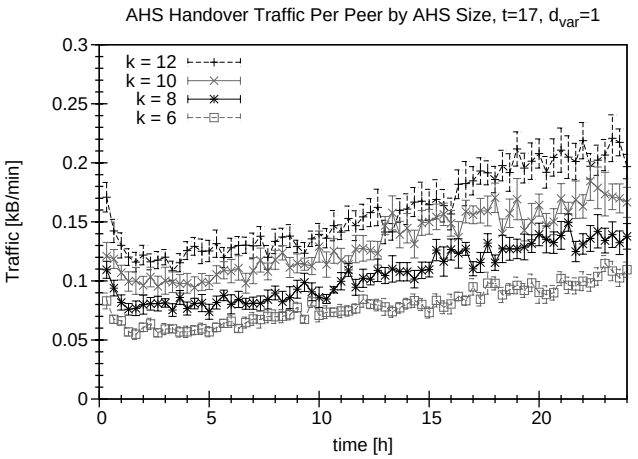
Aggregation Account Handover Mechanism

Account handovers are performed when a peer does a clean logoff. Due to the analytical traffic estimation a linear increase in traffic is expected because with growing account holder set size, more handovers have to be performed at the same churn rate.

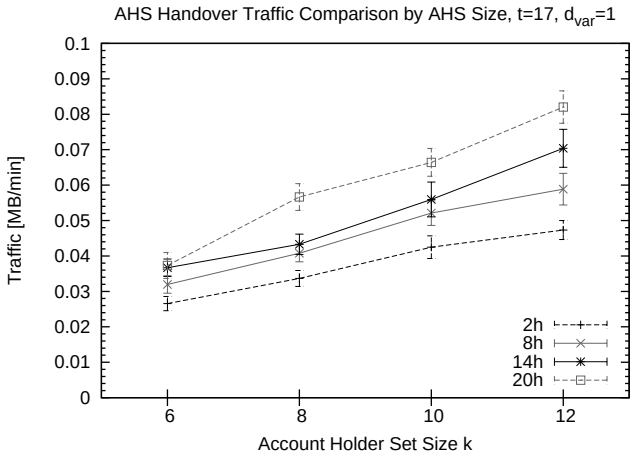
The traffic created by handovers is depicted in Figure 5.33 a). It increases over time, because an increasing number of peers perform handovers. Figure 5.11 shows that the number of peers decreases over time. At the end of the simulation time for $k = 6$ handovers create 0.04 MBytes per minute of



(a) Aggregation account handover traffic by AHS size



(b) Aggregation account handover traffic per peer by AHS size



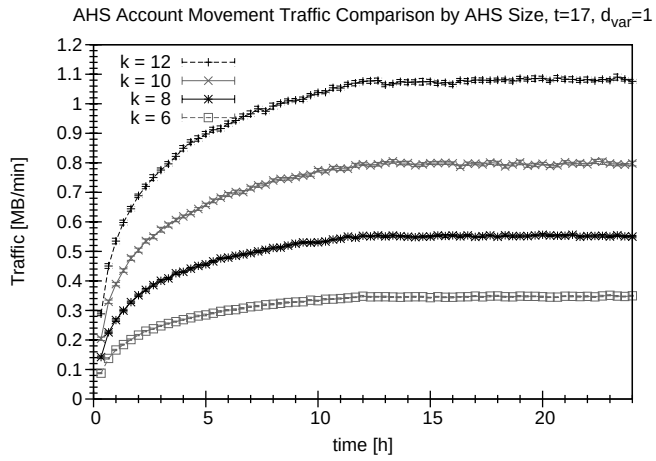
(c) Aggregation account handover traffic development by AHS size

Figure 5.33: Aggregation account handover traffic by AHS size

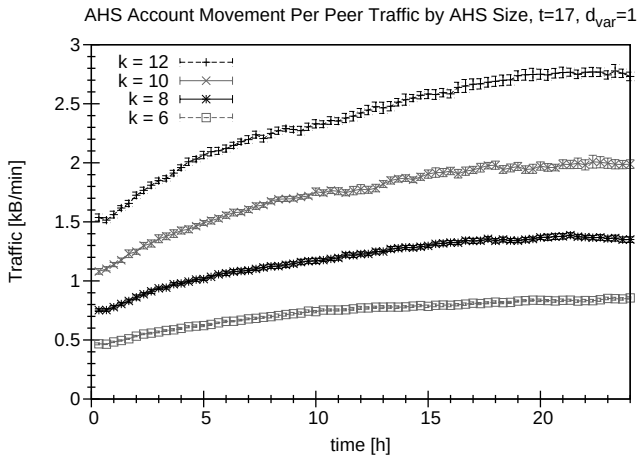
traffic, for $k = 8$ they create 0.06 MBytes of traffic, for $k = 10$ they create 0.07 MBytes of traffic, and for $k = 12$ they create 0.08 MBytes of traffic. Overall handover traffic approximately doubles for a doubled account holder set size. The detailed traffic development over the account holder set size is shown in Figure 5.33 b). The best fit curve results in $V_{AHS\,Handover} = 0.0001k^2 + 0.0044k + 0.0139$ MBytes with a coefficient of determination $R^2 = 0.9997$. The quadratic factor here is negligible. The average handover traffic per peer is for $k = 6$ is 0.11 kBytes per minute, for $k = 8$ is 0.14 kBytes per minute, for $k = 10$ is 0.17 kBytes per minute, for $k = 12$ is 0.21 kBytes per minute at the end of the simulation.

Aggregation Account Movement Mechanism

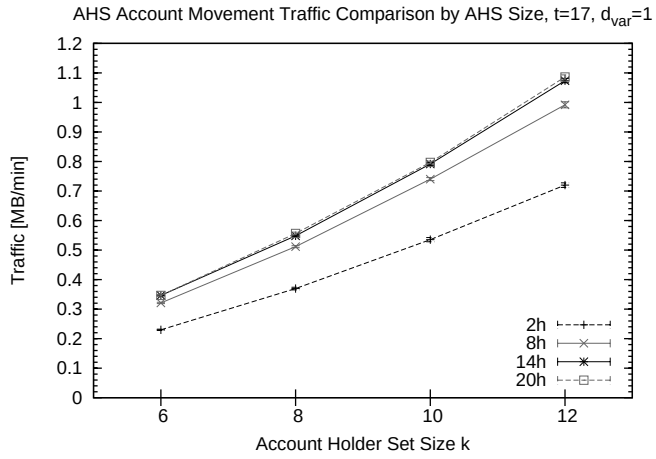
Similar to the previous mechanisms, accounts are moved on demand. The traffic created is similar to that of the account handover traffic, because the mechanisms are executed consecutively. First, the traffic is lower, because the check for account positions starts after a specific period after an account was initialised. Account movement is not executed that often in the beginning. Figure 5.34 shows the traffic generated by the account movement mechanism for the different account holder set sizes.



(a) Aggregation account movement traffic by AHS size



(b) Aggregation account movement traffic per peer by AHS size



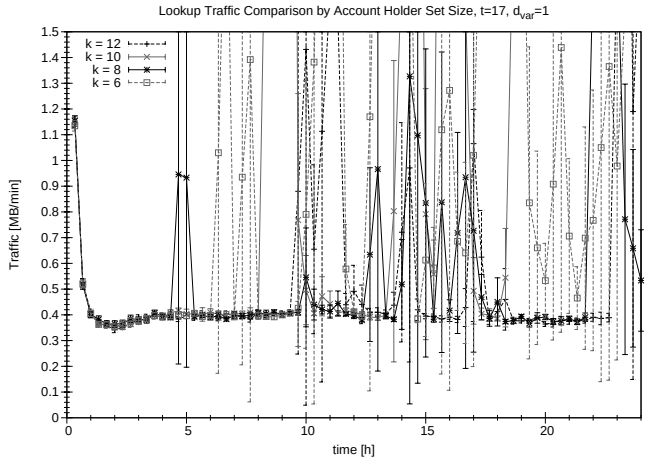
(c) Aggregation account movement traffic development by AHS size

Figure 5.34: Account movement traffic by AHS size

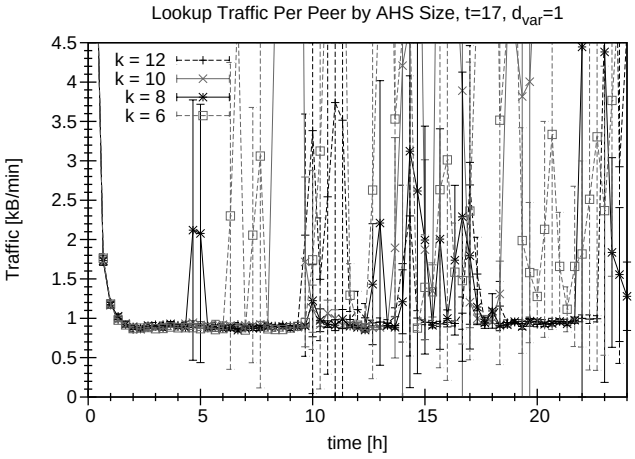
At the end of the simulation the average traffic per minute is 0.35 MBytes for $k = 6$, 0.55 MBytes for $k = 8$, 0.80 MBytes for $k = 10$, and 1.08 MBytes for $k = 12$. Overall, the traffic increased by a factor of approximately 3.1 for a doubled account holder set size. Figure 5.34 b) shows the development of traffic in more detail. The best fit curve for this development results in $V_{AHS\,Movement} = 0.0005k^2 + 0.0301k - 0.0159$ with a coefficient of determination R^2 =1. At the end of the simulation account movement traffic per peer for $k = 6$ is 0.85 kBytes per minute, for $k = 8$ is 1.36 kBytes per minute, for $k = 10$ is 1.99 kBytes per minute, $k = 12$ is 2.76 kBytes per minute.

Lookup Traffic

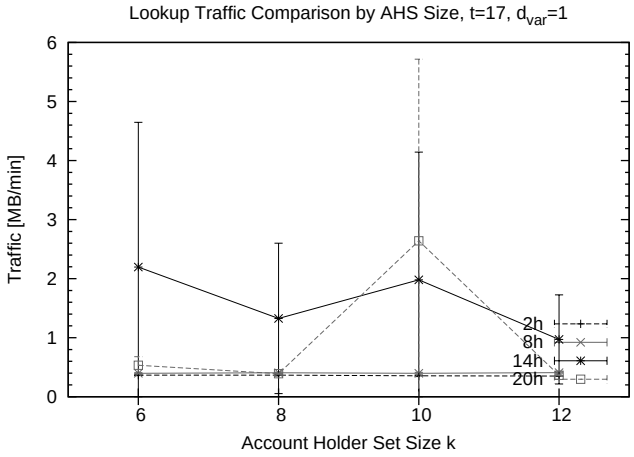
Lookup Traffic is constant, because lookups are performed per account holder set. Figure 5.35 shows the traffic generated by lookups. The volatile traffic is due to the malfunction of Chord. Not considering the traffic generated due to this malfunction, average per minute lookup traffic is approximately 0.38 MBytes system-wide and 0.90 kBytes per peer. Of the lookups approximately 43.5% are created due to system maintenance and 56.5% are created due to transactions.



(a) Lookup traffic by AHS size



(b) Lookup traffic per peer by AHS size



(c) Lookup traffic development by AHS size

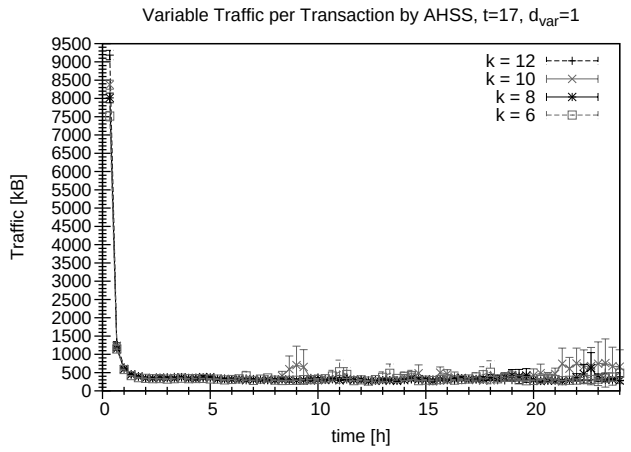
Figure 5.35: Lookup traffic by AHS size

Average Chord maintenance traffic per minute is, over all these experiments, constant at approximately on average 6.67 MBytes system-wide and 15.86 kBytes per peer.

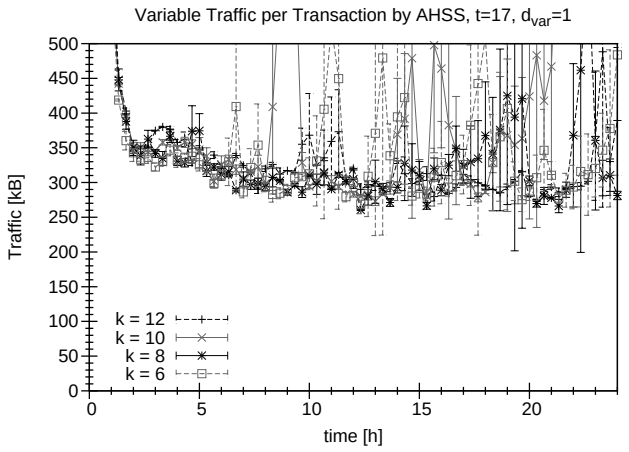
Overall Traffic

Figure 5.36 shows the comparison of the variable traffic per transaction by account holder size. Due to the malfunction of Chord this traffic varies strongly over time. However, variable traffic is almost constant with varying account holder set size. At simulation time from 4 hours, 20 minutes to 6 hours, 20 minutes, the variable traffic is on average 338.10 kBytes for $k = 12$, 336.45 kBytes for $k = 10$, 334.78 kBytes for $k = 8$, and 322.83 kBytes for $k = 6$. However, the variable traffic is decreasing at that time. At a simulation time of 12 hours and 20 minute, where the variation is low, the variable traffic is 290.19 kBytes for $k = 12$, 280.45 kBytes for $k = 10$, 260.68 kBytes for $k = 8$, and 268.53 kBytes for $k = 6$. Fitting a trendline does not lead to meaningful results, as the variations are stronger than the difference in traffic due to the account holder set size. However, it is clear that the development of variable traffic is linear with an increasing account holder set size. This is in accordance with the theoretical results from Section 5.3.

Figure 5.37 shows the average fixed traffic per minute per peer by account holder set size. It is slowly growing over time with increasingly active aggregation accounts. At simulation time from 4 hours, 20

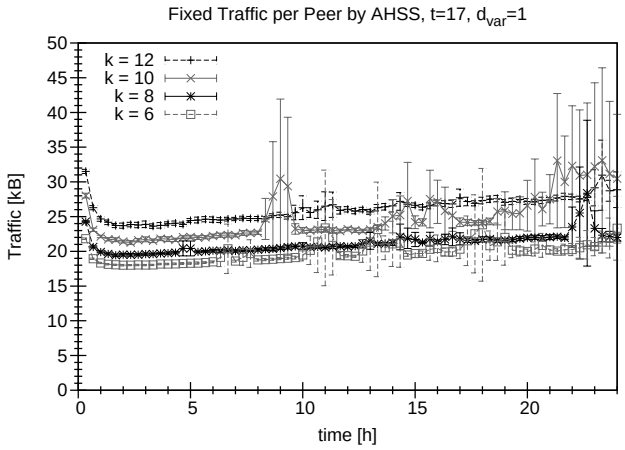


(a) Variable Traffic per transaction by AHS size



(b) Variable traffic per transaction by AHS size (focused)

Figure 5.36: Variable traffic by AHS size



(a) Fixed traffic per peer by AHS size

Figure 5.37: Fixed traffic by AHS size

minutes to 6 hours, 20 minutes fixed traffic is approximately 24.38 kBytes per minute per peer for $k = 12$, 22.13 kBytes for $k = 10$, 20.24 kBytes for $k = 8$, and 18.64 kBytes for $k = 6$. Later in the simulation at 12 hours, 20 minutes, it is approximately 25.80 kBytes per minute per peer for $k = 12$, 23.07 kBytes for $k = 10$, 20.83 kBytes for $k = 8$, and 19.46 kBytes for $k = 6$. Towards the end of the simulation, from 19 hours, 20 minutes to 19 hours, 40 minutes it is approximately 27.33 kBytes per minute per peer for $k = 12$, 25.60 kBytes for $k = 10$, 21.86 kBytes for $k = 8$, and 20.45 kBytes for $k = 6$. On average over these simulation times, fixed traffic results in a best fit curve of $v_{fixed}(k) = 0.0619k^2 - 0.0309k + 17.413$ kBytes per minute per peer with a coefficient of determination $R^2 = 0.9962$. Accordingly, the fixed traffic is increasing with a weak quadratic factor with increased account holder set size. This is in accordance with the theoretical results from Section 5.3.

Summary

The account holder set size experiments show overall a strong increase of the related traffic with an increased account holder set size. These are mainly the account holder set maintenance mechanisms.

Overall account holder set maintenance traffic increases by a factor of 3.06 for a doubled account holder set size. This is because most maintenance mechanisms' traffic grows by a quadratic factor when the account holder set size is increased; still the quadratic factor is smaller than the linear factor. The most traffic within the maintenance mechanisms is created by account locking and account consistency. Both mechanisms have large quadratic factors in the best fit curve for the traffic development. This fits to the analytic results; both mechanism have a message complexity $O(k^2)$. The query and update of aggregation accounts traffic grows almost linearly. It grows only by a very small quadratic factor when the account holder set size is increased.

Aggregation traffic and payment traffic do not vary distinctively for different account holder set sizes. Aggregation is effected the most by failures in the overlay network. It uses lookups heavily to determine the aggregation administrator and the quorum peers.

Account holder set size did not show a relevant influence on peers' upload queue length.

Overall, account holder set maintenance has a stronger influence on the created traffic then the quorum size. Therefore, the professional must pay close attention to the correct configuration of the account holder set size when the token-based accounting scheme is deployed in practise. For the simulated scenarios, system-wide traffic varies between 11.67 MBytes per minute for $k = 6$ and 14.22 MBytes per minute for $k = 12$ in a system of 1000 peers. This is on average, per active peer approximately 25.55 kBytes per minute for $k = 6$ and approximately 32.37 kBytes per minute for $k = 12$. This does not consider the traffic stemming from the malfunction of the Chord overlay. It is to be noted that for the churn used in these simulations an account holder set size of 6 is sufficient.

In conclusion, although the account holder set size k has the strongest influence on the traffic created by the token-based accounting scheme, the generated traffic is small compared to service traffic with only 32.37 kBytes per minute on average per peer. The actual amount of traffic created in a real deployment, even for peers with low bandwidth connections, only consumes a low percentage of the available upload bandwidth.

5.4.3.5 Influence of Churn on Traffic Overhead

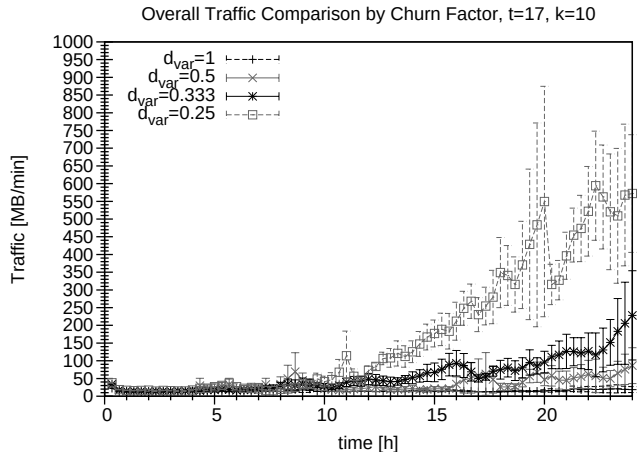
In order to evaluate the scalability of the token-based accounting scheme, two different kinds of experiment variations have been analysed. The first one varies the churn factor d_{var}. The second simulates different sizes of the p2p system, which is presented in the next section.

Total Token-based Accounting Traffic

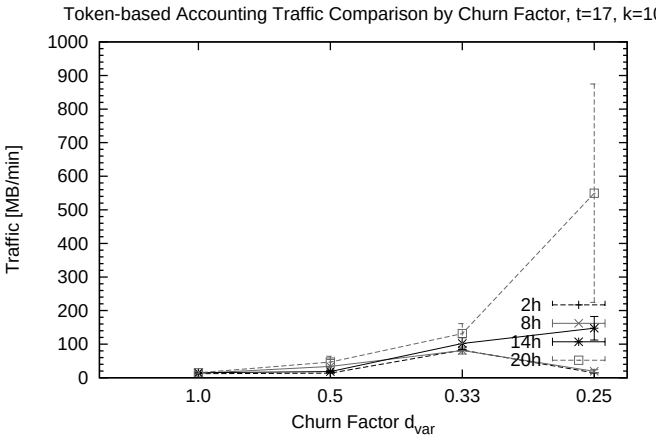
The peers' online and offline times are multiplied by the churn factor d_{var}. For example, with a churn factor of $d_{var} = 0.25$, the online and offline times are quartered.

Figure 5.38 shows the results for the complete token-based accounting traffic. Figure 5.38 a) shows an increase by a factor of approximately 30 from the normal churn model to the churn model with quartered online and offline time of peers. As Figure 5.38 c) shows, at the end of the simulation, approximately 98% of this traffic stems from lookups. This is due to the described malfunction of Chord. This also shows that the Chord protocol requires improvements if it is to be applied in scenarios with strong churn. Obviously, these results are not representative. Therefore, the total token-based accounting traffic has been analysed without considering lookup traffic. Figure 5.39 shows these results.

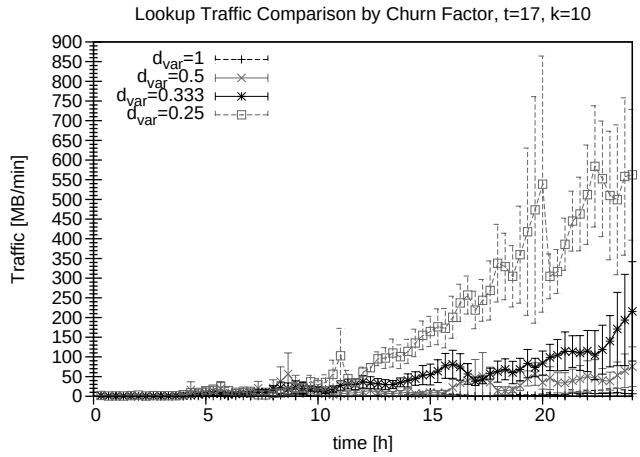
Figure 5.39 a) shows that the traffic at the beginning of the simulation is the higher the stronger the churn is. The reason is that due to the shorter offline time the peers are joining faster. Therefore at the beginning more aggregation accounts have to be created than in scenarios with weaker churn. However, at the end of the simulation total traffic is lower for stronger churn than for weaker churn. The reason is that the number of transactions decreases more quickly over time for stronger churn (see Figure5.39 d). Accordingly, for analysing the influence of churn on the traffic created, transaction dependent traffic and non-transaction dependent traffic have to be analysed separately. It is not meaningful to give a



(a) System-wide TBAS traffic by churn factor

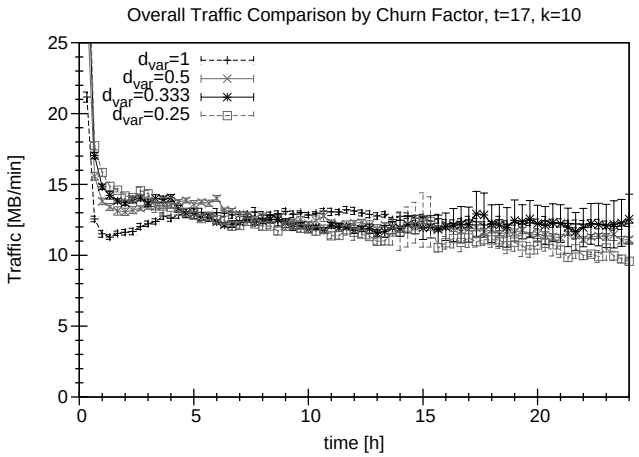


(b) System-wide TBAS traffic development by churn factor

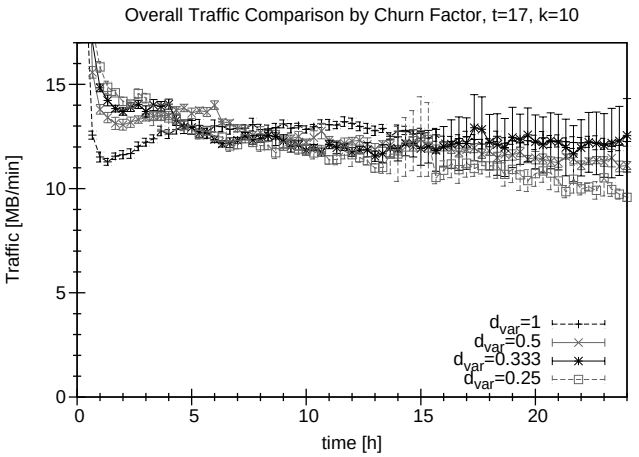


(c) System-wide lookup traffic by churn factor

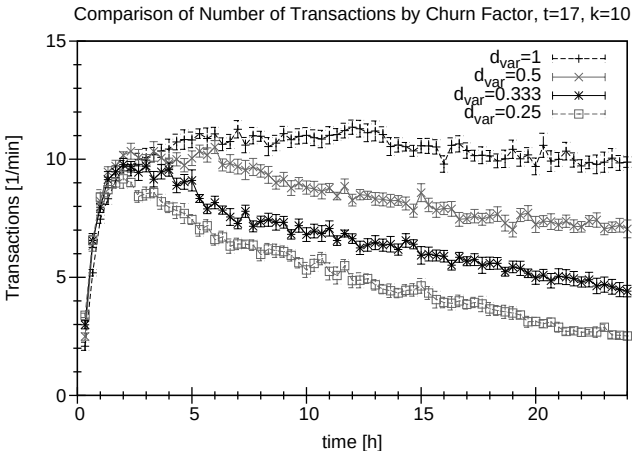
Figure 5.38: Total token-based accounting traffic by churn factor d_{var}



(a) System-wide TBAS traffic without lookup traffic by churn factor



(b) System-wide TBAS traffic without lookup traffic by churn factor (focused)



(c) System-wide number of transactions by churn factor

Figure 5.39: Total token-based accounting traffic without lookup traffic by churn factor d_{var}

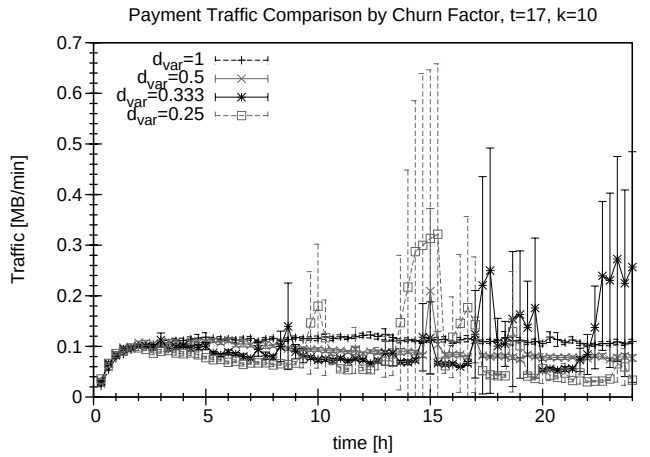
general traffic development by churn factor, because the results are distorted by the different number of transactions per experiment.

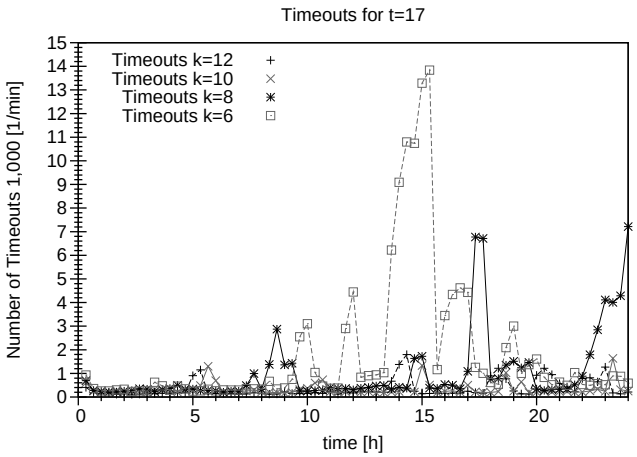
The reason for the stronger decrease of transaction traffic for a stronger churn is the frequency users' request a service. If a peer's next request time is after its logoff time, the request is discarded. This happens more often in higher churn situations.

Payment Traffic

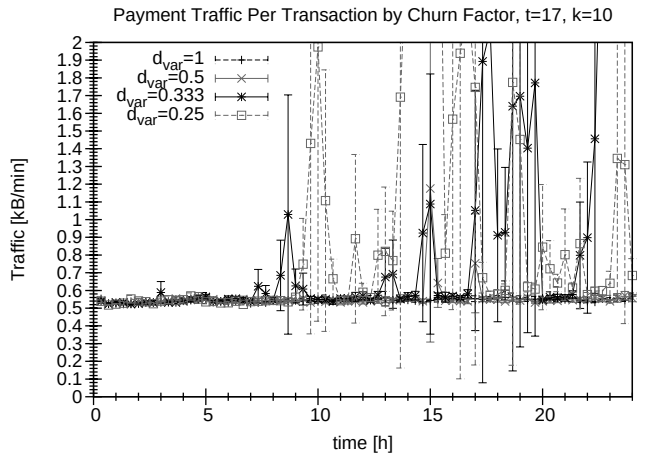
The malfunction of Chord led to increased payment traffic in the account holder set size experiments. This was in correlation with the number of timeouts happening within the simulation. These effects are most easily observed within the churn experiments. Payment requires updating aggregation accounts. When the lookup of an aggregation account fails during payment due to the malfunction of chord, further timeouts trigger sending further messages in order to request the status of the payment process. This way, the cascaded timeouts lead to a traffic increase that can be observed in Figure 5.40.

Figure 5.40 a) shows the system-wide average payment traffic per minute by churn factor. As discussed above, this is not meaningful, as the number of transactions decrease over time. The average payment





(a) System-wide payment traffic by churn factor



(b) System-wide payment traffic per transaction by churn factor

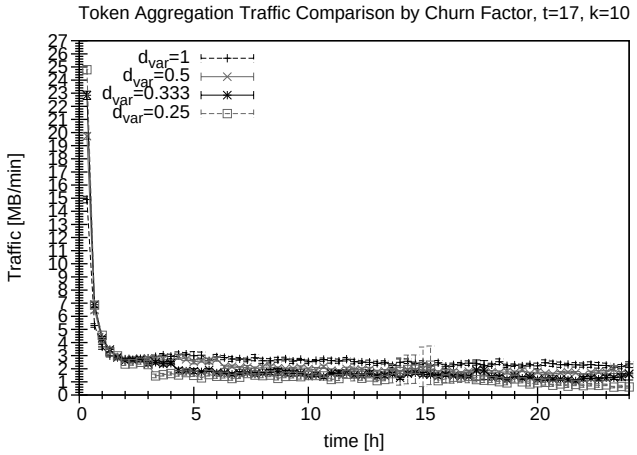
Figure 5.40: Payment traffic by churn factor

traffic per transaction is depicted in Figure 5.40 c). It shows that payment traffic per transaction does not differ with varying churn factor, and is on average 10.9 kBytes per transaction.

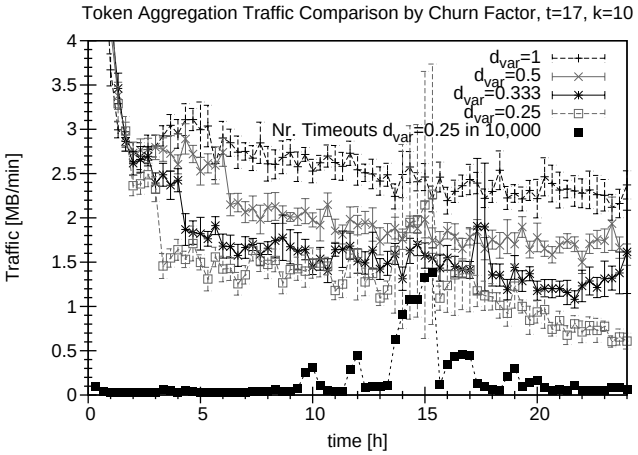
Token Aggregation Traffic

Token aggregation traffic depends on the number of transactions performed in the system. Figure 5.41 a) shows the average system-wide token aggregation traffic per minute. The stronger the churn, the higher the initial token aggregation traffic, due to the larger number of initial joins that require the creation of an aggregation account and the starting tokens. Figure 5.41 b) shows the system-wide token aggregation traffic focused on the time after the start-up phase. Here outliers for $d_{var} = 0.25$ can be observed. This is due to timeouts that require resending messages. Figure 5.41 c) depicts the average token aggregation traffic per transaction, as well as the timeouts, where the upper four series depict the average token aggregation traffic per transaction and the lower four series depict the average number of timeouts. The presentation of the individual churn factors is the same as in Figure 5.41 a) and b).

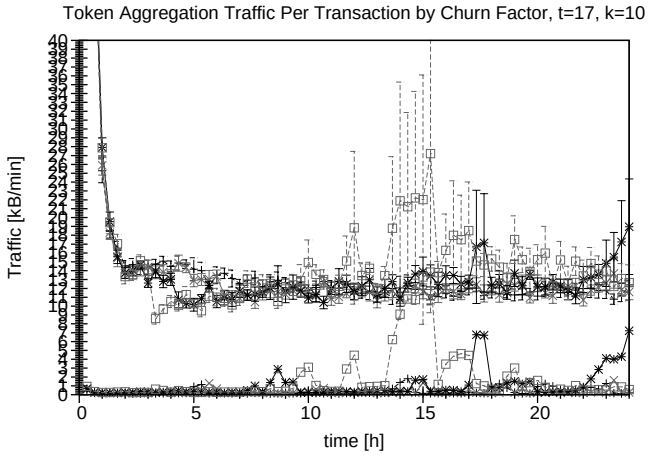
After the start-up phase, token aggregation traffic per transaction is constant and does not differ by churn factor. The timeouts are responsible for the observable differences, as can be seen in Figure 5.41 c). In an overlay network working correctly, these timeouts would not occur. Observing token aggregation



(a) System-wide token aggregation traffic by churn factor



(b) System-wide token aggregation traffic by churn factor (focused)



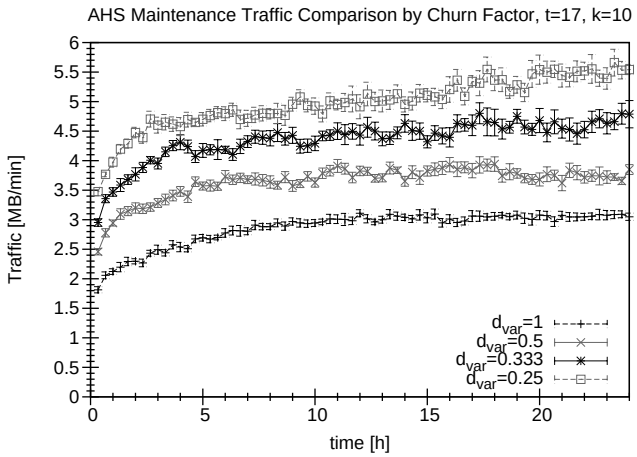
(c) Token aggregation traffic per transaction by churn factor

Figure 5.41: Token aggregation traffic by churn factor

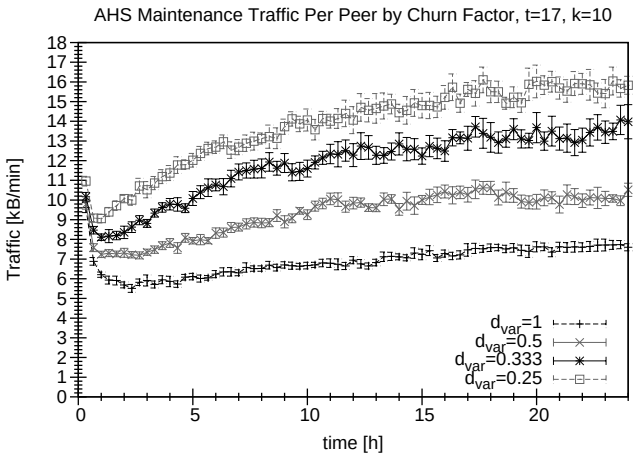
traffic at points of time where the influence of timeouts is minimal, the average token aggregation traffic for all churn factors is on average approximately 239 kBytes per transaction. Note that this traffic is distributed among all peers participating in the token aggregation process.

All Account Holder Set Maintenance Mechanisms

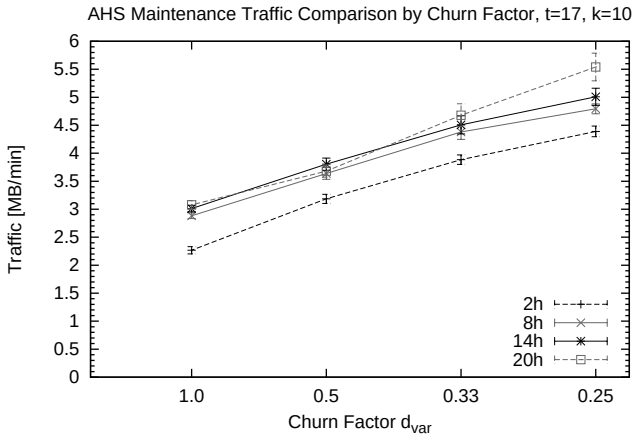
Figure 5.42 shows the traffic created by all account holder set maintenance mechanisms. From the beginning of the simulation, the maintenance traffic increases and then flattens at the end of the simulation, after all aggregation accounts have been initialised. Account holder set maintenance traffic increases with stronger churn. At the end of the simulation time, system-wide account holder set maintenance traffic on average for $d_{var} = 1$ is 3.08 MBytes, for $d_{var} = 0.5$ is 3.74 MBytes, for $d_{var} = 0.333$ is 4.72 MBytes, and for $d_{var} = 0.25$ is 5.50 MBytes. This per peer on average for $d_{var} = 1$ is 7.70 kBytes, for $d_{var} = 0.5$ is 10.15 kBytes, for $d_{var} = 0.333$ is 13.73 kBytes, and for $d_{var} = 0.25$ is 15.68 kBytes, which is shown in Figure 5.42 b). The best fit curve for the traffic development at 20 hours simulation time is $V_{AHS\,Maintenance} = 0.0482 d_{var}^2 + 0.5771 d_{var} + 2.4294$ MBytes with a coefficient of determination $R^2 = 0.9962$.



(a) System-wide maintenance traffic by churn factor

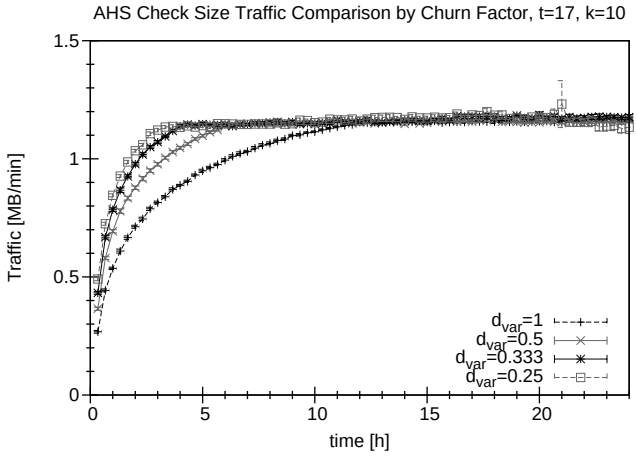


(b) Maintenance traffic per peer by churn factor

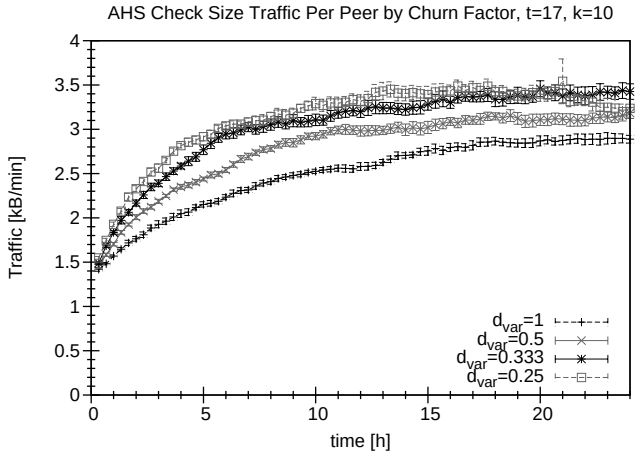


(c) System-wide maintenance traffic development

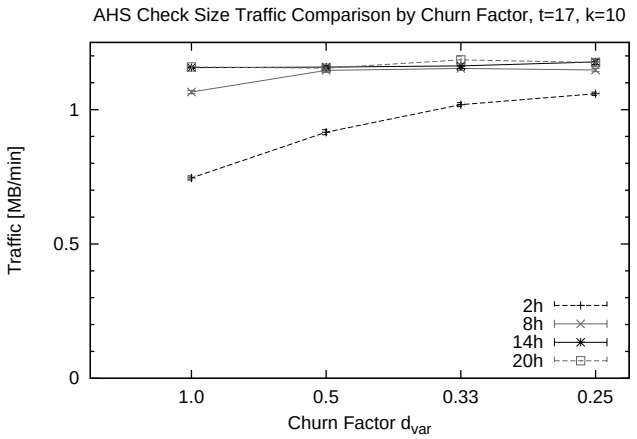
Figure 5.42: Account holder set maintenance by churn factor



(a) System-wide check size traffic by churn factor



(b) Check size traffic per peer by churn factor



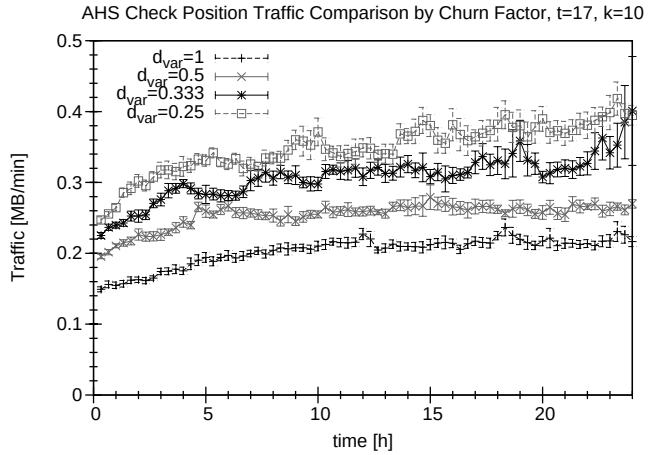
(c) Check size traffic development by churn factor

Figure 5.43: Account holder set check size traffic by churn factor

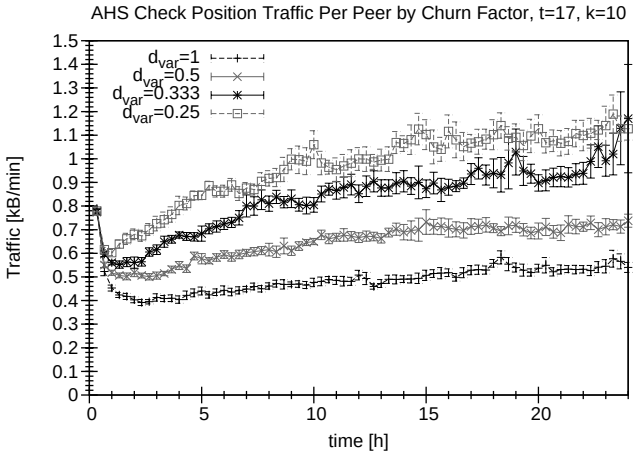
Account Holder Set Check Size Mechanism

Similar to the overall account holder set maintenance traffic, the account holder check size traffic increases with the number of initialised aggregation accounts and then flattens. At the end of the simulation the traffic decreases slightly for $d_{var} = 0.25$. This is caused by the strong churn, which decreases the average account holder set size. Furthermore, the check size mechanism is called with a constant frequency, which is independent of churn. The result is a reduced traffic that can be observed at the end of the simulation time for the $d_{var} = 0.25$ scenario.

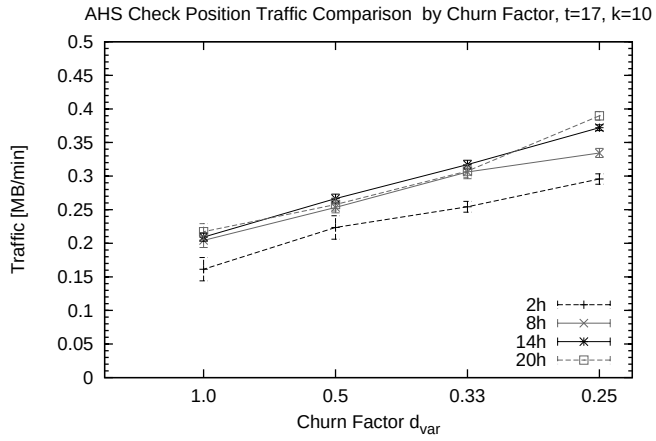
Check account holder set size traffic is highest a approximately from 17 hours to 18 hours simulation time. At that time the average system-wide the traffic for $d_{var} = 1$ is 1.16 MBytes per minute, for $d_{var} = 0.5$ is also 1.16 MBytes per minute, for $d_{var} = 0.333$ is 1.18 MBytes per minute, and for $d_{var} = 0.25$ is 1.19 MBytes per minute. This per peer on average for $d_{var} = 1$ is 2.85 kBytes per minute, for $d_{var} = 0.5$ is 3.10 kBytes per minute, for $d_{var} = 0.333$ is 3.39 kBytes per minute, and for $d_{var} = 0.25$ is 3.40 kBytes per minute. The best fit curve for the system-wide traffic is $V_{AHS\,CheckSize} = 0.001d_{var}^2 + 0.006d_{var} + 1.1495$ MBytes with a coefficient of determination $R^2 = 0.99772$.



(a) System-wide check position traffic by churn factor



(b) Check position traffic per peer by churn factor

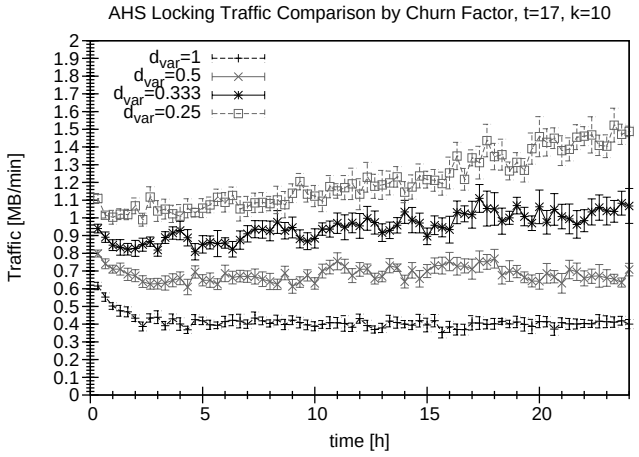


(c) Check position traffic development by churn factor

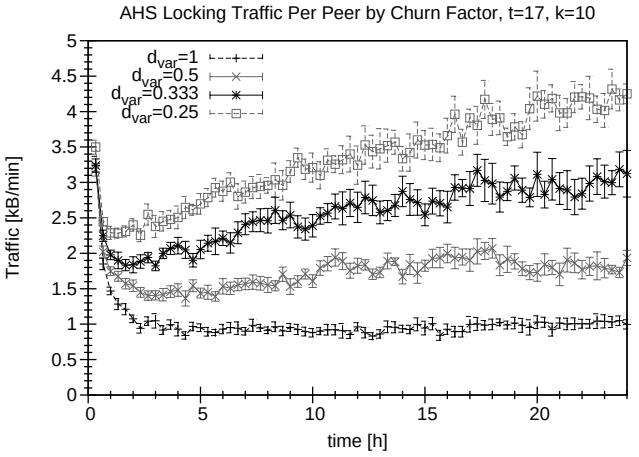
Figure 5.44: Account holder set check position traffic by churn factor

Account Holder Set Check Position Mechanism

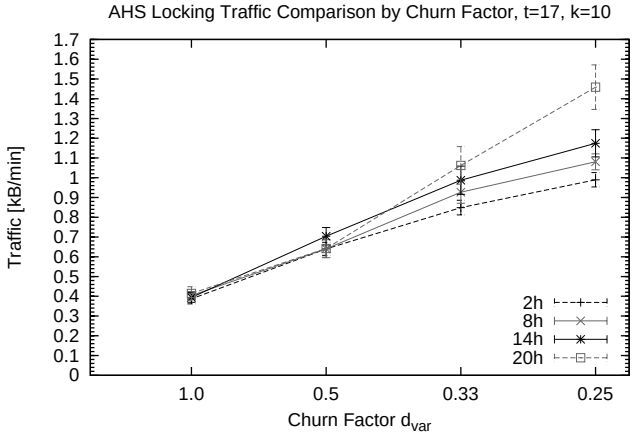
The system-wide account holder set check position mechanism traffic is lowest at the beginning of the simulation. In increases until the end of the simulation, where it flattens for $d_{var} = 1$ and $d_{var} = 0.5$. For the other two churn factors, the traffic is still increasing at the end of the simulation. The reason is the strong churn that requires many repositions. However, the repositions depend on lookups, which are less successful at the end of the simulation due to the malfunction of Chord (see Figure 5.38 c)). Therefore, timeouts occur during checking and determining new account holder set positions, which results in increased traffic. Between simulation time of 20 hours and 22 hours the system-wide check position traffic on average for $d_{var} = 1$ is 0.21 MBytes per minute, for $d_{var} = 0.5$ is 0.26 MBytes per minute, for $d_{var} = 0.333$ is 0.32 MBytes per minute, and for $d_{var} = 0.25$ is 0.38 MBytes per minute. This is per peer on average for $d_{var} = 1$ is 0.53 kBytes per minute, for $d_{var} = 0.5$ is 0.70 kBytes per minute, for $d_{var} = 0.333$ is 0.92 kBytes per minute, and for $d_{var} = 0.25$ is 1.09 kBytes per minute. The best fit curve for the system-wide traffic results in $V_{AHS\,CheckPosition} = 0.0029 d_{var}^2 + 0.0396 d_{var} + 0.172$ MBytes with a coefficient of determination $R^2 = 0.9999$.



(a) System-wide AHS locking traffic by churn factor



(b) AHS locking traffic per peer by churn factor



(c) AHS locking traffic development by churn factor

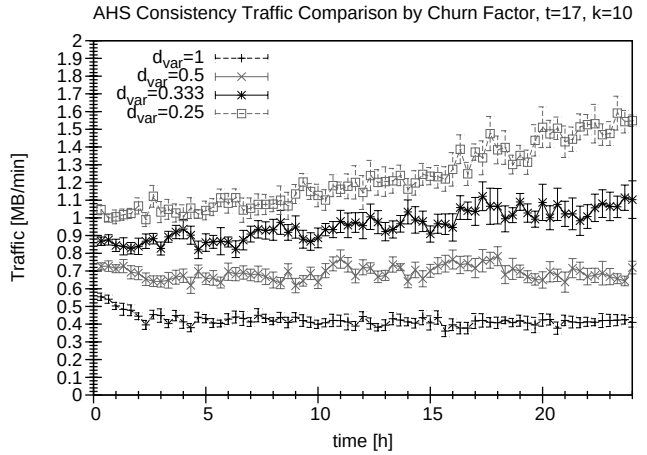
Figure 5.45: Locking aggregation account traffic by churn factor

Lock Aggregation Account Mechanism

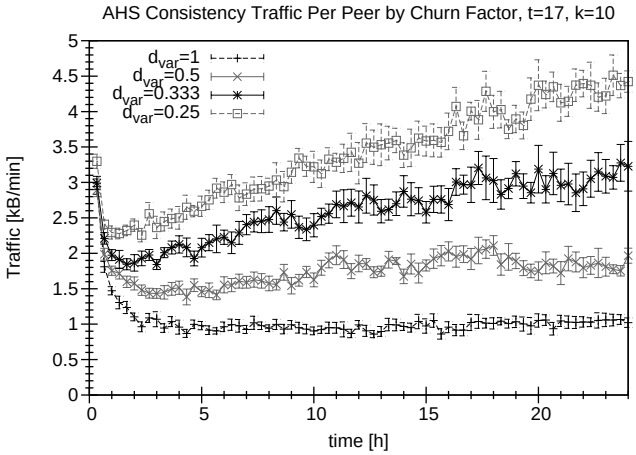
The aggregation account locking mechanism traffic is flat after the start-up phase for $d_{var} = 1$ and $d_{var} = 0.5$. For $d_{var} = 0.333$ the traffic increases slowly, and for $d_{var} = 0.25$ the traffic increases distinctly at the end of the simulation time. At the end of the simulation time, system-wide aggregation account locking traffic on average for $d_{var} = 1$ is 0.41 MBytes per minute, for $d_{var} = 0.5$ is 0.67 MBytes per minute, for $d_{var} = 0.333$ is 1.04 MBytes per minute, and for $d_{var} = 0.25$ 1.46 MBytes per minute. This per peer on average for $d_{var} = 1$ is 1.02 kBytes per minute, for $d_{var} = 0.5$ is 1.80 kBytes per minute, for $d_{var} = 0.333$ is 3.03 kBytes per minute, and for $d_{var} = 0.25$ is 4.17 kBytes per minute. The best fit curve for the system-wide traffic results in $V_{AHS\,Locking} = 0.0404d_{var}^2 + 0.1511d_{var} + 0.2139$ MBytes with a coefficient of determination $R^2 = 0.9995$.

Aggregation Account Consistency Mechanism

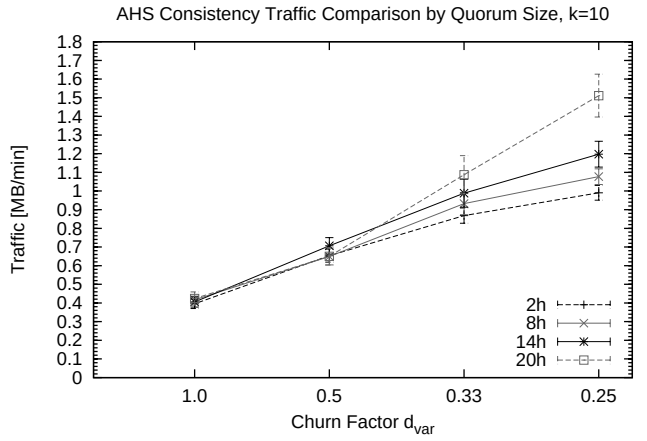
The aggregation account consistency mechanism traffic develops similarly to that of the aggregation account locking mechanism. At the end of the simulation time, system-wide aggregation account consistency traffic on average for $d_{var} = 1$ is 0.42 MBytes per minute, for $d_{var} = 0.5$ is 0.67 MBytes per minute, for $d_{var} = 0.333$ is 1.04 MBytes per minute, and for $d_{var} = 0.25$ is 1.53 MBytes per minute. This per peer on average for $d_{var} = 1$ is 1.05 kBytes per minute, for $d_{var} = 0.5$ is 1.83 kBytes per minute,



(a) System-wide AHS consistency traffic by churn factor



(b) AHS consistency traffic per peer by churn factor



(c) AHS consistency traffic development by churn factor

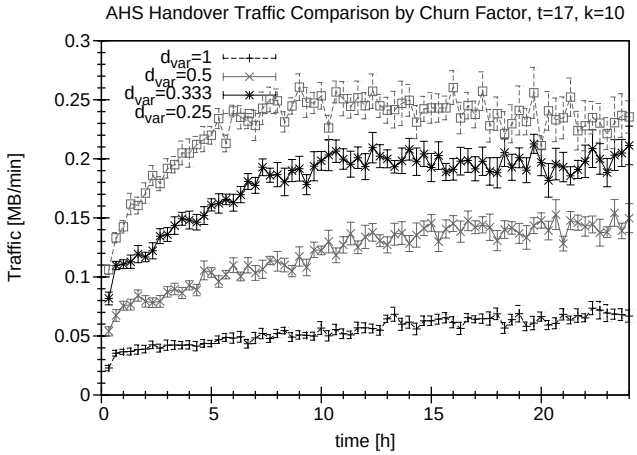
Figure 5.46: Aggregation account consistency traffic by churn factor

for $d_{var} = 0.333$ is 3.11 kBytes per minute, and for $d_{var} = 0.25$ is 4.36 kBytes per minute. The best fit curve for the system-wide traffic results in $V_{AHS\,Consistency} = 0.05d_{var}^2 + 0.122d_{var} + 0.2419$ MBytes with a coefficient of determination $R^2 = 0.9996$.

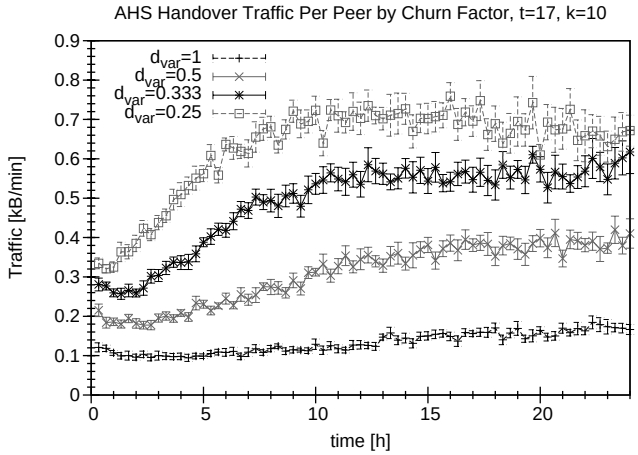
Aggregation Account Handover Mechanism

Aggregation account handover traffic first increases with the increasing number of initialised aggregation accounts. Then the traffic flattens. For $d_{var} = 0.25$ the traffic decreases at the end of the simulation time, similar to the check size mechanism. Failing peers reduce the average account holder set size. As churn is constant over the simulation time, the handover mechanism is executed less frequently when the average account holder set size is reduced. Thus, after the end of the start-up phase where joins were more frequent, the account handover traffic decreases slowly again until the average account holder set size stabilises. Figure 5.47 d) shows the average account holder set size by churn factor.

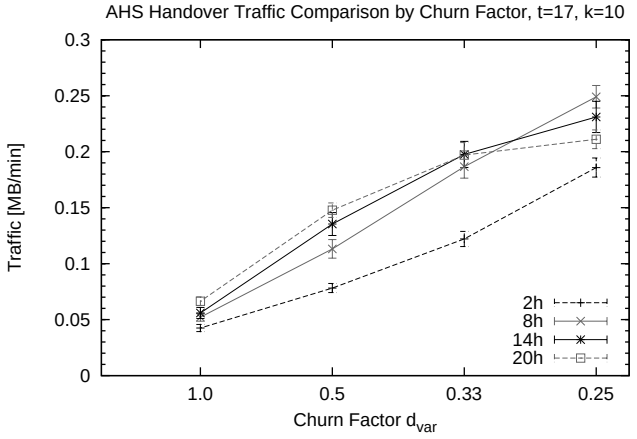
At the end of the simulation time system-wide aggregation account handover traffic on average for $d_{var} = 1$ is 0.07 MBytes per minute, for $d_{var} = 0.5$ is 0.14 MBytes per minute, for $d_{var} = 0.333$ is 0.20 MBytes per minute, and for $d_{var} = 0.25$ is 0.23 MBytes per minute. This per peer on average for $d_{var} = 1$



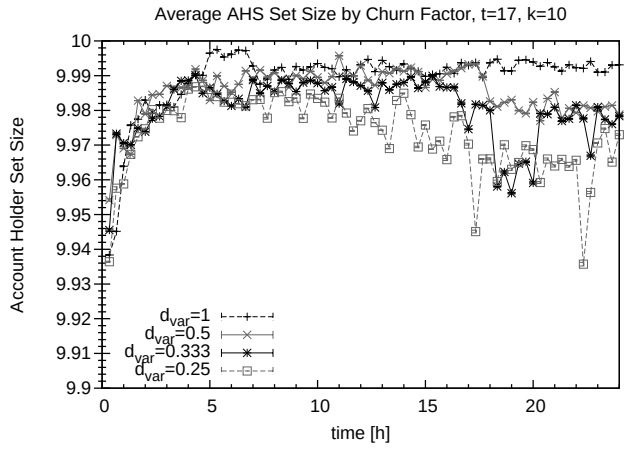
(a) System-wide AHS handover traffic by churn factor



(b) AHS handover traffic per peer by churn factor



(c) AHS handover traffic development by churn factor



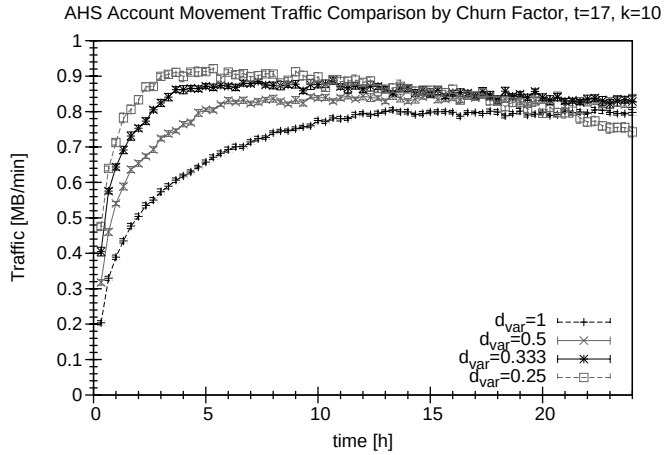
(d) Average account holder set size by churn factor

Figure 5.47: Aggregation account handover traffic by churn factor

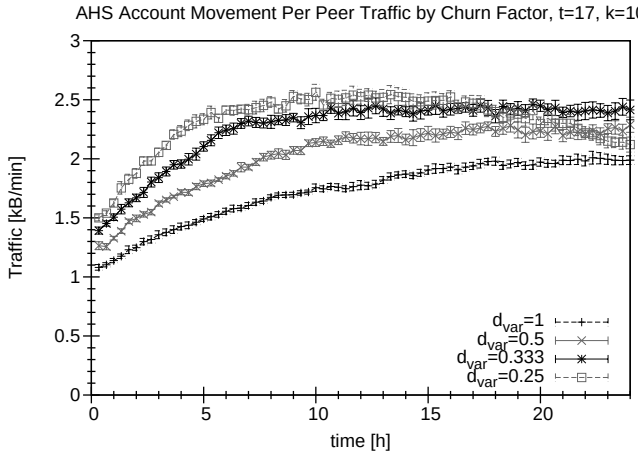
is 0.17 kBytes per minute, for $d_{var} = 0.5$ is 0.39 kBytes per minute, for $d_{var} = 0.333$ is 0.59 kBytes per minute, and for $d_{var} = 0.25$ is 0.66 kBytes per minute. The best fit curve for the system-wide traffic results in $V_{AHS\,Handover} = -0.0114 d_{var}^2 - 0.1114 d_{var} + 0.0316$ MBytes with a coefficient of determination $R^2 = 0.9994$.

Aggregation Account Movement Mechanism

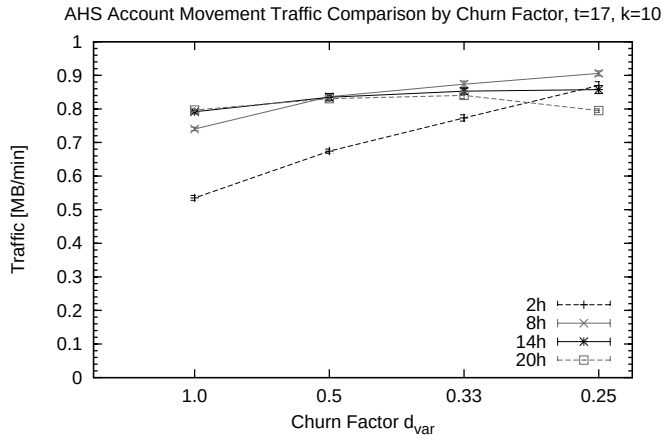
Aggregation account movement traffic first increases with the increasing number of initialised aggregation accounts. Then the traffic flattens. For $d_{var} = 0.25$ the traffic decreases at the end of the simulation time, similar to the check size and the handover mechanism. The reason herefor is that over time the average account holder set size is slowly decreasing. Also for $d_{var} = 0.333$, the traffic decreases slightly at the end of the simulation time. At that time system-wide aggregation account movement traffic on average for $d_{var} = 1$ is 0.80 MBytes per minute, for $d_{var} = 0.5$ is 0.83 MBytes per minute, for $d_{var} = 0.333$ is 0.83 MBytes per minute, and for $d_{var} = 0.25$ is 0.76 MBytes per minute. This per peer on average for $d_{var} = 1$ is 1.99 kBytes per minute, for $d_{var} = 0.5$ is 2.24 kBytes per minute, for $d_{var} = 0.333$ is 2.41 kBytes per minute, and for $d_{var} = 0.25$ is 2.16 kBytes per minute. The best fit curve



(a) System-wide AHS movement traffic by churn factor



(b) AHS movement traffic per peer by churn factor



(c) AHS movement traffic development by churn factor

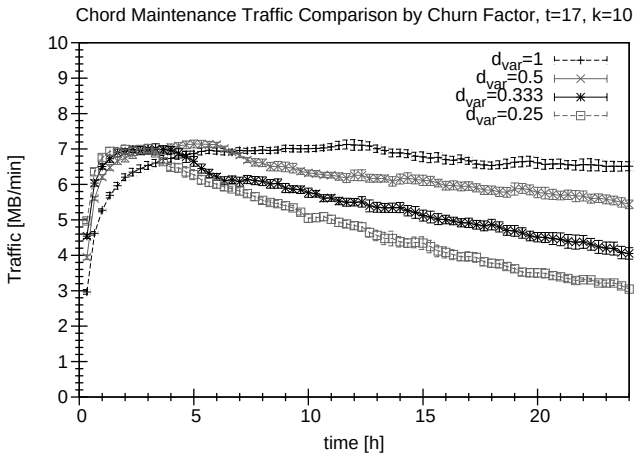
Figure 5.48: Aggregation account movement traffic by churn factor

for the system-wide traffic results in $V_{AHS\,Movement} = -0.0257d_{var}^2 + 0.1168d_{var} + 0.7048$ MBytes with a coefficient of determination $R^2 = 0.9678$.

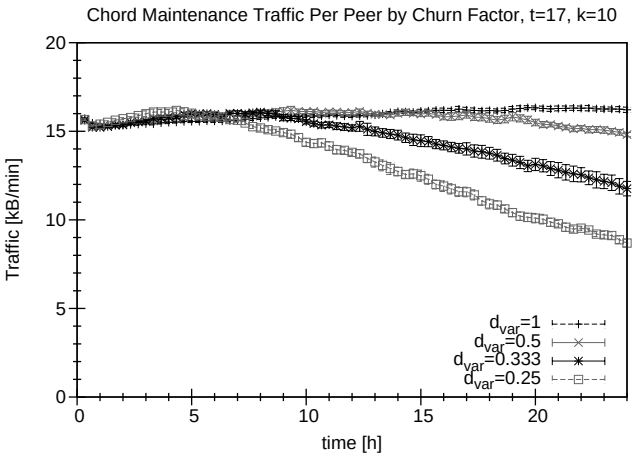
Chord Maintenance

Chord Maintenance traffic first increases sharply at the beginning of the simulation and then decreases again. This is due to the join process, because peers join the first time after half the offline time. Therefore, at the beginning more peers join than on average will be online after the system stabilises. However, chord maintenance traffic still decreases after the average number of active peers has stabilised. The reason is in the Chord stabilisation implementation that does not take further actions when a timeout happens. If a stabilisation query message cannot be delivered to its destination, the destination peer is simply added to the sender's list of non-available peers. The stabilisation query message is not re-sent to a different peer. The missing response messages reduce the Chord maintenance traffic.

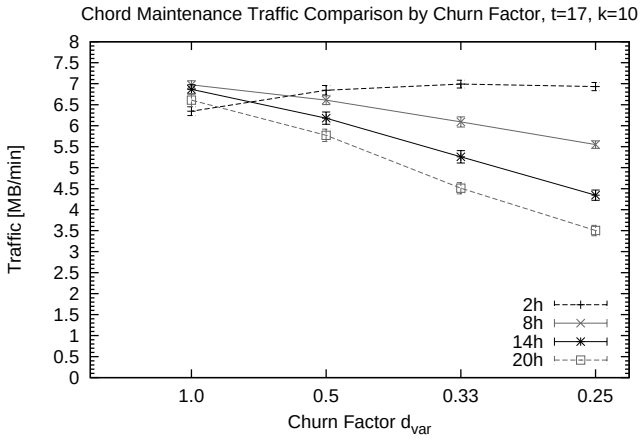
At the end of the simulation time system-wide Chord maintenance traffic on average for $d_{var} = 1$ is 6.51 MBytes per minute, for $d_{var} = 0.5$ is 5.47 MBytes per minute, for $d_{var} = 0.333$ is 4.07 MBytes per minute, and for $d_{var} = 0.25$ is 3.09 MBytes per minute. This per peer on average for $d_{var} = 1$ is 16.23



(a) System-wide Chord maintenance traffic by churn factor



(b) Chord maintenance traffic per peer by churn factor



(c) Chord maintenance traffic development by churn factor

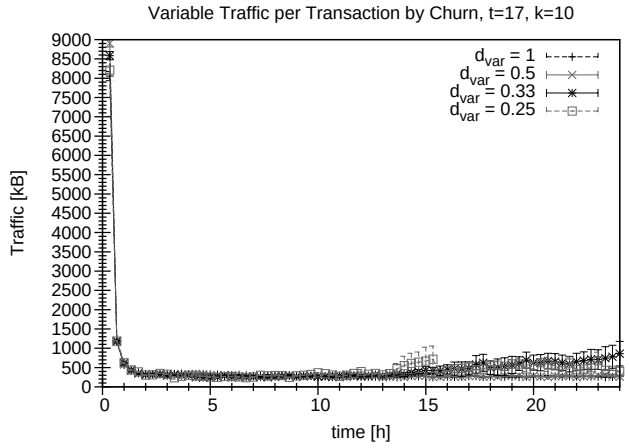
Figure 5.49: Chord maintenance traffic by churn factor

kBytes per minute, for $d_{var} = 0.5$ is 14.88 kBytes per minute, for $d_{var} = 0.333$ is 11.83 kBytes per minute, and for $d_{var} = 0.25$ is 8.79 kBytes per minute. The best fit curve for the system-wide traffic results in $V_{AHS\,ChordMaintenance} = 0.0181 d_{var}2 - 1.2567 d_{var} + 7.7891$ MBytes with a coefficient of determination $R^2 = 0.9956$.

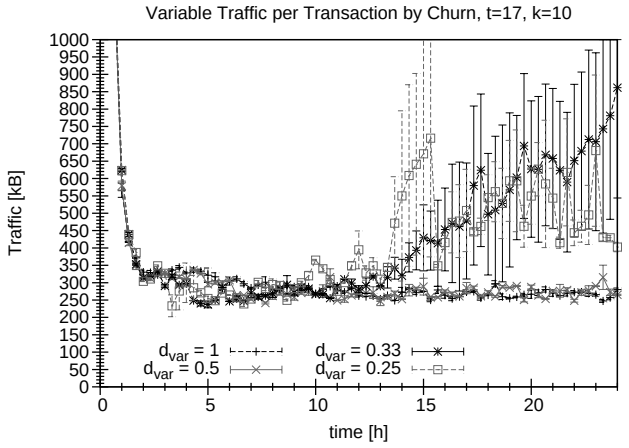
Overall Traffic

For analysing the influence of churn on the variable and fixed traffic, lookup traffic was not considered due to the intensive malfunctions of Chord.

Figure 5.50 shows the comparison of the variable traffic per transaction by churn factor. Here, the volatile traffic stems from the high number of timeouts that increased aggregation and payment traffic (see Figures 5.41 and 5.40). When these timeouts do not happen, variable traffic is approximately constant over the churn factors. This is in accordance with the theoretical results from Section 5.3. Then, variable traffic is on average approximately 273.22 kBytes per transaction.

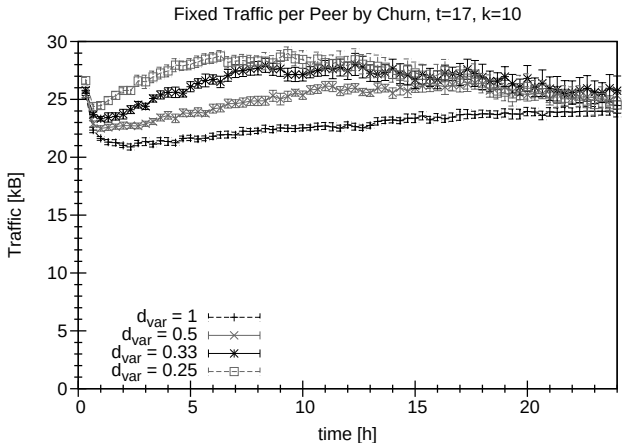


(a) Variable traffic per transaction by churn factor



(b) Variable traffic per transaction by churn factor (focused)

Figure 5.50: Variable traffic by churn factor



(a) Fixed traffic per peer by churn factor

Figure 5.51: Fixed traffic by churn factor

Figure 5.51 shows the average fixed traffic per minute per peer by churn factor. The development of the fixed traffic over different churn factors differs over time, because of the Chord maintenance traffic and the account holder set maintenance traffic that decreases over time for strong churn.

At simulation time from 4 hours, 20 minutes to 6 hours, 20 minutes fixed traffic is approximately 24.38 kBytes per minute per peer for $d_{var} = 1$, 21.64 kBytes for $d_{var} = 0.5$, 23.88 kBytes for $d_{var} = 26.21$, and 28.28 kBytes for $d_{var} = 0.25$. This results in a best fit curve of $v_{fixed}(d_{var}) = -0.042d_{var}^2 + 2.4344d_{var} + 19.231$ kBytes per peer per minute with a coefficient of determination $R^2 = 0.9997$. This is slightly less-than-linear development. During the last two hour in the simulation, it is approximately 23.07 kBytes per minute per peer for $d_{var} = 1$, 25.15 kBytes for $d_{var} = 0.5$, 25.75 kBytes for $d_{var} = 0.33$, and 24.78 kBytes for $d_{var} = 0.25$. This results in a best fit curve of $v_{fixed}(d_{var}) = -0.5497d_{var}^2 + 3.0631d_{var} + 21.378$ kBytes per minute per peer with a coefficient of determination $R^2 = 0.9691$. This would probably change with a different Chord implementation that is more reactive to missing successors and fingers.

Summary

As expected, churn has no influence on the payment traffic and token aggregation traffic created per transaction. Overall, the number of transactions is reduced slightly with stronger churn, which is due to the service request model, which is kept constant.

Churn influences the account holder set maintenance traffic. The stronger the churn, the more often the account holder set maintenance mechanisms have to be executed. Overall, per peer account holder set maintenance traffic increases from 7.70 kBytes per minute to 15.68 kBytes per minute when peers' online time and offline time is quartered. This is an increase by a factor of 2.03. The best fit curve for the per peer account holder set maintenance traffic is $v_{AHS\,Maintenance} = -0.036d_{var}^2 + 0.3466d_{var} - 0.1432$ kBytes with a coefficient of determination $R^2 = 0.9961$. Accordingly, the traffic increase has a quadratic factor that is approximately half as strong as the linear factor.

The churn experiments show some unexpected behaviour of the traffic created. First of all, the malfunction of Chord, which could be observed for $d_{var} = 1$, leads to a traffic increase in lookups that for $d_{var} = 0.25$ at the end of the simulation time consumes 1.55 MBytes of traffic per minute per peer. This traffic would completely congest a peer's upload queue. The reason why the malfunction happens more frequently is the Chord maintenance traffic, which decreases significantly for strong churn over the simulation time. For $d_{var} = 0.25$ the Chord maintenance traffic is only 44% of it maximum at the end of the simulation time. When stabilisation does not succeed, the existing malfunctions in the Chord ring will not be repaired. The lookup could be reduced, e.g., by limiting the lifetime of lookup messages to a reasonable value. However, for strong churn scenarios, the Chord protocol has to be improved and should also consider its predecessors and successors for routing lookup messages, not only the fingers.

The further unexpected result is the reduced traffic at the end of the simulation time for some account holder set maintenance mechanisms. With strong churn, the average account holder set size is reduced. Thus, mechanisms that create traffic that depends on the account holder set size might create less traffic with stronger churn than with normal churn. For example, the check size mechanism is executed per account holder set with a constant frequency independent of churn; the amount of traffic created depends on the current set size. Therefore, the traffic is reduced with strong churn. Furthermore, with a reduced average account holder set size, the total number of account replicas existing in the system is reduced. Therefore, account handovers and account movement traffic are also reduced. Overall, the effect of this traffic reduction is compensated by the other account holder set maintenance mechanisms, which create more traffic when the average account holder set size is decreased in order to repair the account holder sets.

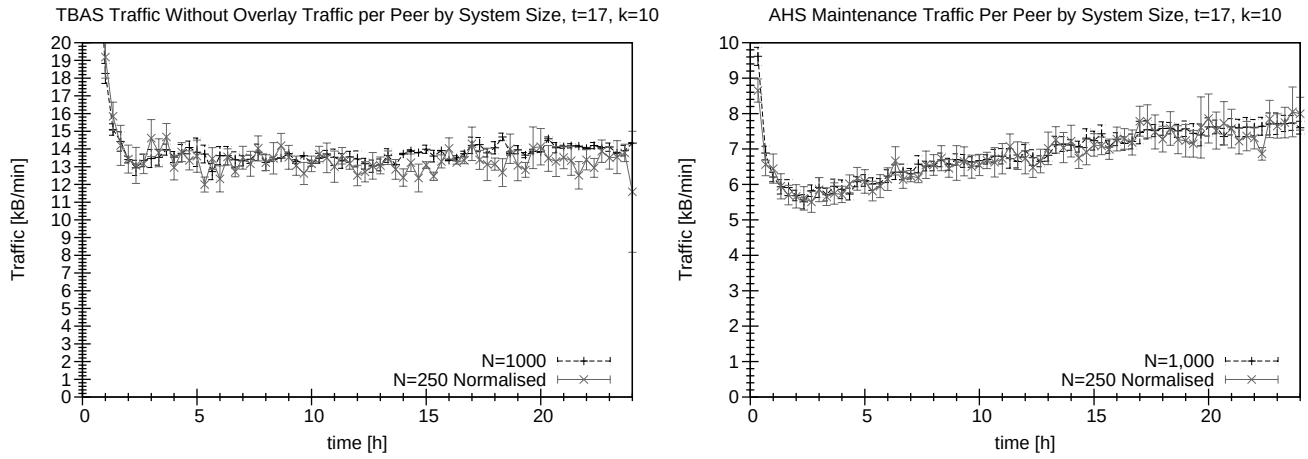
Although the Chord overlay had struggles operating under the strong churn, the number of transactions and token aggregations performed did not suffer. Furthermore, the maintenance traffic created grew less than linearly. Therefore, the robustness of the token-based accounting scheme against failures could be demonstrated.

5.4.3.6 Influence of System Size on Traffic Overhead

In analysing the token-based accounting mechanisms, it was found that there are no protocols that are influenced by the system size. Accordingly, it can be expected that payment and token aggregation traffic should be constant per transaction, and the account holder set maintenance traffic should be constant per peer. Lookup traffic should be reduced, because the lookup messages travel fewer hops on average for a smaller system size.Chord maintenance traffic should be reduced, because fewer fingers have to be maintained.

In conclusion, when comparing the traffic created by the token-based accounting scheme for different network sizes, it should be considered without the overlay traffic, because this distorts the results.

Observing the percentage of active peers, for $N = 250$ the percentage was slightly higher (for $N = 250$, 44.3% of peers where active on average, for $N = 1000$, 43.3% of peers were online on average). The percentage of online peers influences the created traffic per peer, because the number of online peers



(a) Token-based accounting traffic per peer by system size

(b) AHS check size traffic per peer by system size

Figure 5.52: Token-based accounting traffic and AHS maintenance traffic per peer by system size

influences the number of transactions performed in the system, which influences the amount of payment and transaction traffic. Furthermore, account holder set maintenance traffic is created by initialised aggregation accounts. Accordingly, in order not to distort the per peer traffic results, the results for $N = 250$ were normalised to the same percentage of online peers as for $N = 1000$. Figure 5.52 a) shows the resulting average token-based accounting traffic per peer per minute. There is a very small difference in the generated traffic. The average token-based accounting traffic per peer is 13.70 kBytes per minute for $N = 1000$, and 13.28 kBytes per minute for $N = 250$. This is a difference of 3.14% on average after the start-up phase. This difference stems from the different number of transactions executed per peer on average, which are after the start-up phase for $N = 250$ on average 3.29% less than for $N = 1000$.

Figure 5.52 b) presents the average account holder set maintenance traffic per peer per minute; here the traffic is very similar so that the difference cannot be determined with statistical confidence.

Figure 5.53 a) shows the average payment per transaction. The payment traffic for $N = 1000$ is more even than the traffic for $N = 250$, because the average is calculated over a larger number of transactions. At the end of the simulation (between 21 hours and 23 hours of simulation time) payment traffic for $N = 1000$ is approximately 0.55 kBytes per transaction, which is slightly higher than payment traffic for $N = 250$, which is 0.53 kBytes per transaction. This difference is 4.5% of the payment traffic for $N = 1000$.

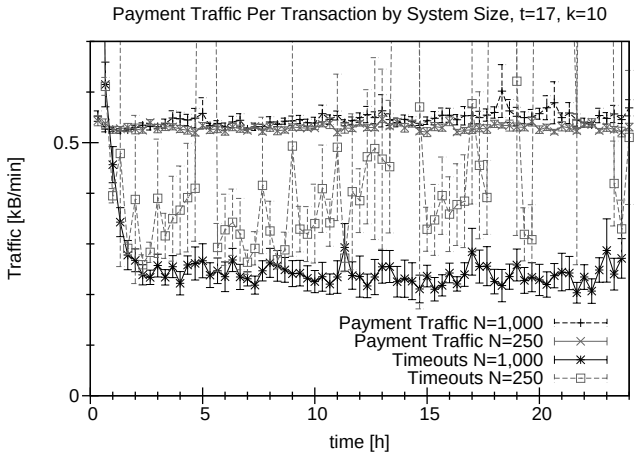
Figure 5.53 b) shows the average token aggregation traffic per transaction. Here, a difference is hard to determine and cannot be concluded with statistical confidence.

In summary, it can be concluded that with an increased system size, only overlay traffic increases per peer. The token-based accounting traffic is constant when the system size varies.

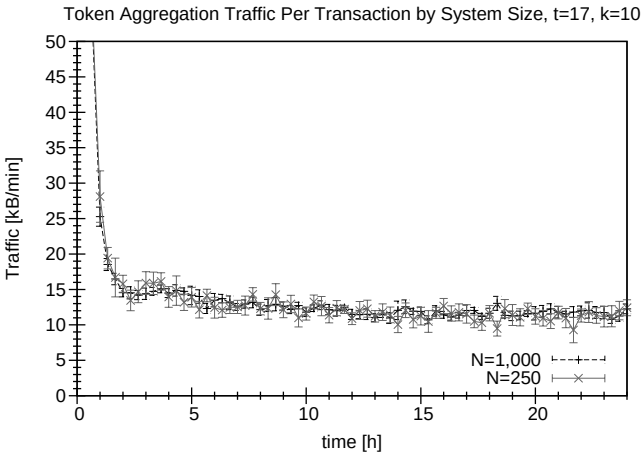
5.4.4 Summary

The simulation of the token-based accounting scheme analysed the influence of the quorum size t, the account holder set size k, churn, and system size on the traffic generated by the scheme. As the base experiment a quorum size of $t = 17$, an account holder set size $k = 10$, a churn factor $d_{var} = 1$, and a system size of $N = 1000$ were selected. When the influence of one parameter was analysed, the other parameters remained constant (ceteris paribus).

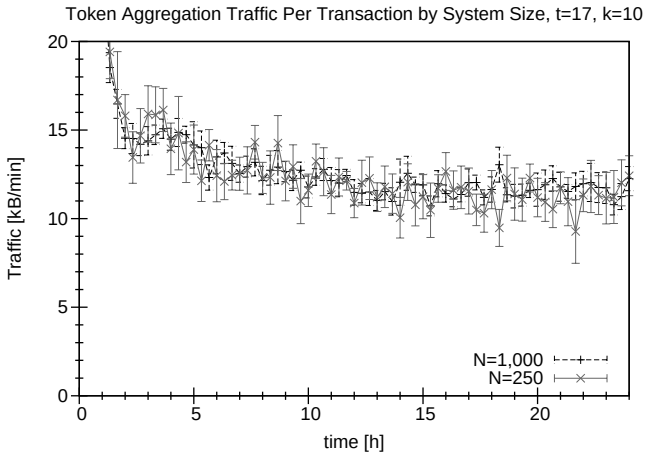
The results for the base experiment showed that the token-based accounting scheme creates approximately 13.4 MBytes traffic per minute on average system-wide. Per active peer this is on average



(a) Payment traffic per transaction by system size



(b) Token aggregation traffic per transaction by system size



(c) Token aggregation traffic per transaction by system size (focused)

Figure 5.53: Payment and token aggregation traffic per transaction by system size

approximately 30.30 kBytes per minute. Within the simulated music sharing scenario, the token-based accounting scheme created approximately 17.86% of the traffic and 82.14% was created from simulated services. The traffic breakdown showed that approximately 293.58 kBytes of traffic is generated per transaction, which is distributed over the peers participating in the transaction and the corresponding token aggregation. Furthermore, account holder set maintenance requires, on average, approximately 22.77 kBytes of traffic per peer per minute. Analysing the peers' load by observing the upload queues showed that most of the time (97.85%) peers have a load less than 1% of their upload capacity and in 99.35% of time, peers have a load less than 10% of their upload capacity, which was set to 128 kBit/sec. 0.65% of the time trusted peers have to bear a higher load of approximately 14 seconds queue length. Accordingly, it is important to select trusted peers according to their upload capacity.

Variation of the quorum size only affected the traffic created by token aggregation. It increases nearly linearly from approximately 6.45 kBytes per minute per active peer for $t = 7$ to approximately 15.21 kBytes per minute per active peer. Furthermore, token aggregation requires lookups, which increase in number with an increase quorum size. For lookups with $t = 7$, 13.64 kBytes of traffic are created per peer per minute on average and with $t = 17$, 17.89 kBytes per minute per peer. The maximum upload queue lengths happen during token aggregation. The approximate maximum queue length for $t = 7$ is 3 seconds, for $t = 17$ is 14 seconds. The increase develops with a quadratic factor. Thus, the careful selection of trusted peers is especially important for larger quorum sizes.

The variation of the account holder set size has only minimal influence on payment and token aggregation traffic. Payment traffic per transaction increases from 0.24 kBytes on average for $k = 6$ to 0.27 kBytes on average for $k = 12$. Similarly, token aggregation traffic per transaction increases from 277.44 kBytes on average for $k = 6$ to 289.32 kBytes on average for $k = 12$. As expected from the analytical results, the account holder set size influences mainly account holder set maintenance traffic. Account holder set maintenance traffic per peer per minute increases from 3.35 kBytes on average for $k = 6$ to 10.74 kBytes on average for $k = 12$. The account locking and the account consistency mechanisms show especially strong growth over the account holder set size , because traffic grows with a distinct quadratic factor with an increase account holder set size. The largest proportion of account holder maintenance traffic is created by the check-size mechanism and the account movement mechanism. The account holder set size does not have a relevant impact on the peer's upload queue lengths.

The variation of churn also mainly influences the account holder set maintenance traffic. For payment and token aggregation traffic, an influence can hardly be noticed. Maintenance traffic is subject to two influences. One is that the maintenance mechanisms e.g., account handover and account movement are executed more frequently. The other is that the average quorum size is reduced due to churn. Therefore, most maintenance mechanisms generate less traffic per execution. Overall, the more frequent distribution has the stronger influence. Maintenance traffic increases from 7.70 kBytes per peer per minute on average for $d_{var} = 1$ to 15.68 kBytes per peer per minute on average for for $d_{var} = 0.25$. Furthermore, Churn showed that the Chord overlay does not perform well for higher churns. Malfunctions that could already be observed in basic churn ($d_{var} = 1$), happened so that frequently that lookup traffic grew exponentially. Therefore, the churn experiments had to be analysed without regarding Chord lookup traffic. Furthermore, this did not allow an analysis of the peers' upload queues, as lookups could not be filtered out afterwards. In relation to the Chord lookup traffic, the Chord maintenance traffic decreases with stronger churn. The reason is in Chord's stabilisation protocol that does not take any special actions when lookups fail but adds the failed peer to the list of unavailable peers. Thus, it can be concluded that the Chord protocol requires improvements when running it is systems with reasonable churn.

Overall, the churn simulations proved the token-based accounting scheme's robustness. Peers could always perform transactions and swap tokens even under strong churn.

During the evaluation of the token-based accounting scheme, several issues arose that led to further findings not directly related to the token-based accounting scheme. The simulation contains three layers: an overlay layer using Chord, the token-based accounting layer, and the application layer containing the user model including churn. Simulating such a system requires that the lower layers are fully operational,

otherwise it distorts the results of the upper layers. The existing implementation of Chord was according protocols presented in (SMK+01, SMLN+03). The first simulation results under churn showed the described malfunction of Chord in 17 - 32 occurrences. Due to failed lookups timeouts were triggered on the upper layers. Often, these timeouts required repeat lookups that again, did not succeed. This resulted overall in traffic that was too high and unrealistic. Therefore, Chord was improved in two iterations under the requirement to retain the core Chord protocols. Still, as the experiments varying churn show, Chord does not work well.

In conclusion, evaluating p2p mechanisms on top of a p2p overlay requires that the Chord overlay is working correctly. The published protocols are not sufficient to re-engineer Chord in a sufficient way for that purpose. Furthermore, p2p overlays have to be evaluated carefully and in long term simulations, simulating one day or longer. The mentioned malfunction of Chord usually happened after 12 hours simulated time. Much more attention has to be paid to unsuccessful messages in the overlay layer, which have effects on the upper layers. When evaluating a pure lookup scenario, lost lookups seem not to have any impact, because they do not influence the functionality of the overlay network. However, having further mechanisms relying on the lookup results, failed lookups here led to a considerably increased amount of traffic. Therefore, carefully tested overlay networks are a core requirement for evaluating the mechanisms on top of them.

In conclusion to this section, the token-based accounting scheme creates on average approximately 30.31 kBytes of traffic per peer per minute in the simulated music sharing scenario. This is composed of variable system-wide traffic per transaction of 293.58 kBytes and a constant traffic per peer per minute of 22.77 kBytes. Therefore, the token-based accounting scheme is useful for all application scenarios where bandwidth is not extremely scarce (which could be the case in emergency management situations for example (BKM08b, BKM08a, BSPM08)). It could be shown that the costs as well as its robustness make the token-based accounting scheme very well suited to typical application scenarios.

In order to guarantee trustworthy operations of the token-based accounting scheme it requires the application of proactive secret sharing in order to ensure the secrecy of the system-wide private key. This is analysed in the following section.

5.5 Simulation of Key Management Traffic

In this section, the simulation results of proactive secret sharing mechanisms which can be applied to ensure the secrecy of the system-wide private key are presented. These mechanisms have been intensively discussed in Sections 4.5 and 4.6. In this section, the results from the analytical traffic assessment and the results from the simulation are compared.

In order to simulate key management, two parts of proactive secret sharing, namely the update phase and the recovery phase were implemented in PeerfactSim.KOM (Mul). The objective of this simulation was to assess the traffic created when all trusted peers receive an updated key share of the system-wide private key. Key management takes place in the separate trusted peer overlay network. Therefore, the simulation of system key maintenance was performed separately from the simulations of the remaining token-based accounting scheme operations.

From the analytical evaluation in Section 4.6 it was concluded that limited update and self-initialisation strategy known from URSA (LL00, LKZ+04) is the only strategy that can be applied in a distributed autonomous system, because it does not require the knowledge of all trusted peers' IDs. After presenting the simulation setup, the executed experiments are described. Then the simulation results will be discussed.

5.5.1 Simulation Setup

Within the simulation of the URSA update and self initialisation strategy each trusted peer has the same functionality. Accordingly, there is only one main class responsible for fulfilling the tasks required by this strategy. The implementation concept of the trusted peer communication as depicted in Figure 5.54.

Update Phase

The responsible peer starts the update process "$UpdateAgent$". So, at the beginning of the simulation, all peers join and the overlay network stabilises. When this process is completed, the $UpdateAgent$ chooses a peer and starts the update and self initialisation process. The update process is started by the method $update()$ within the $URSASelfInitUpdater$ class. It initialises the lookup of a proximity group, which is executing the update. The lookup is performed by the class $DiscoverProximity$. When the discover process is completed, the method $getUpdates()$ is used to look up a time in the local update history, and it returns the update phase iteration. Now, the method $sendUpdateRequest()$ is used to ask all the trusted peers within the proximity group if they want to participate in this update-process. At the receiving trusted peer, this request is processed within the method $handleUpdateRequest()$. A peer will participate if it is in the correct update iteration and if it is not within the dead period before a peer leaves d_{dead}. The signal $handleUpdateRequestReply()$ receives the response. When all peers for a proximity group are found the method $foundProximityToUpdate()$ is called.

A random polynomial is created and for all peers in the proximity group the update shares are computed. [2] The peer will now update itself with its share using the method $updateMySelf()$. Also it distributes the respective update shares to all peers in the proximity group. The method $handleUpdateShareMessage()$ within the $BaseUpdater$ receives this message and adds the update share to the peers share and destroys the old share.

Self Initialisation Phase

When a peer has updated its share successfully, it checks if it should now start to recover remaining trusted peers. This is performed within the method $amIinTheLookupGroup()$. If so, it starts a new discovery job finding a recovery group that can update its successor or predecessor peer, depending on the direction the recovery process runs along the DHT ring. When the group is found, the peers are asked if they are willing to participate in a recovery using the method $sendRequestForRecovery()$. The method $handleRecoveryRequest()$ receives this message. The decision if the peer is willing to participate is made like in the update phase. The response to this message is received by the method $handleRecoveryRequestReply()$. Now, the required recovery information, like the recovery group members, is sent to the recovery group by the method $sendToHelpers()$. This message is received by the method $handleInformToRecover()$. Within this method the cryptographic mechanisms are performed in order to create the partial shares required for a recovery. A recovery message is sent to each peer that is to be recovered by this recovery group. At the peers to be recovered, these messages are collected by the method $handleRecover()$ and when all messages are received, the method $doRecover()$ recovers the share for this peer.

5.5.2 Experiments

When the proximity group is found the update shares have to be distributed among the peers in this group. This traffic is fully deterministic, because only direct communication between the group members is applied. Therefore, the general concept of the experiment is to perform the update using only one update polynomial by one peer. This allows an exact analysis of the created traffic that can be easily extrapolated to the use of more update polynomials.

[2] All parts of the simulation concerning cryptographic operations are not depicted in Figure 5.54 for the sake of clarity.

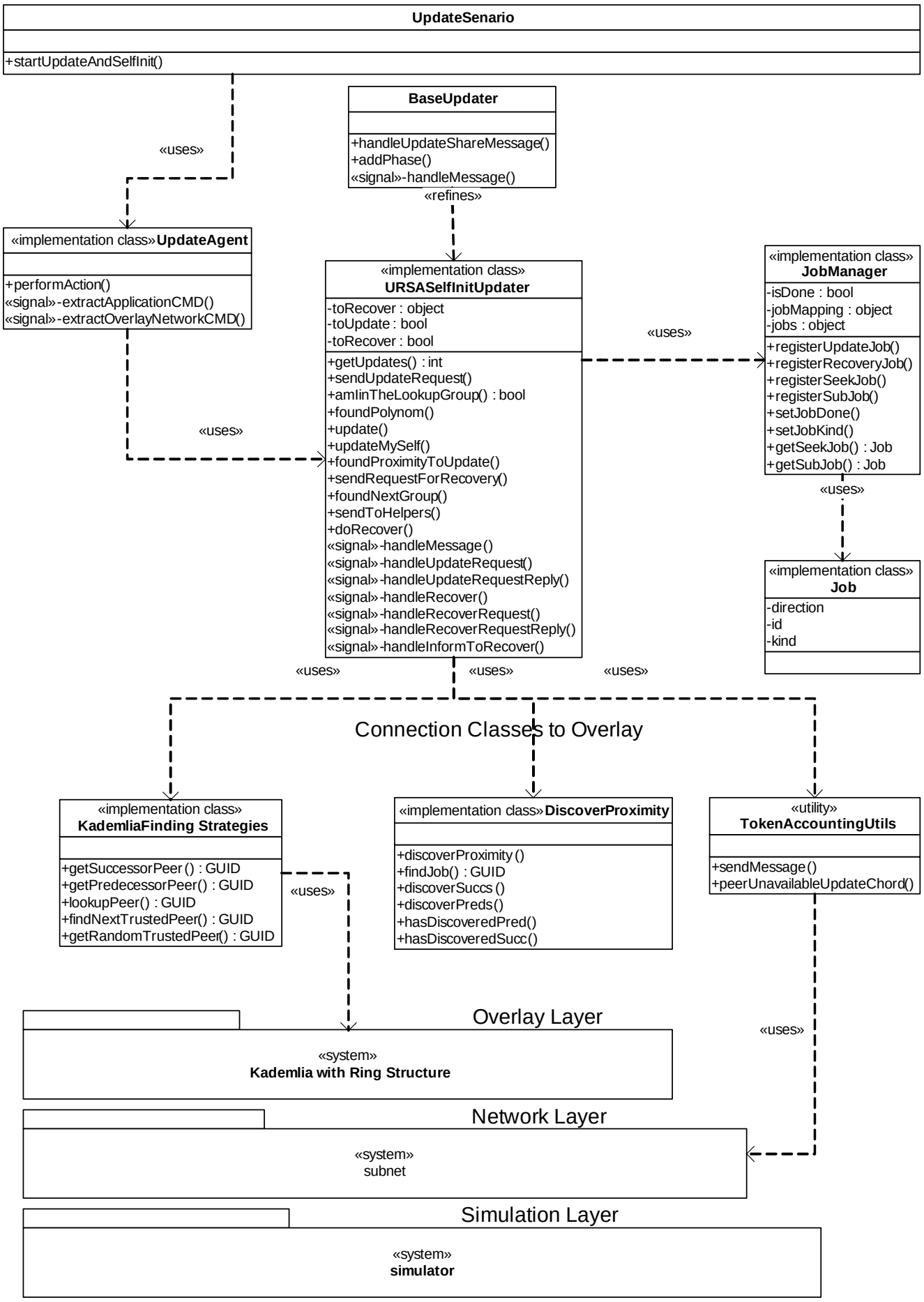


Figure 5.54: URSA update simulation implementation concept

Table 5.7: Resulting Quorum Sizes t for System-key Maintenance Simulations

t	$L_1 = 99.99\%$			$L_2 = 99.999\%$		
T/p_g	0.50	0.70	0.90	0.50	0.70	0.90
100	13	8	4	15	9	5
500	14	8	4	17	10	5
1000	14	8	4	17	10	5
2000	14	8	4	17	10	5

Legend: L : Trust level, T : Number of trusted peers, p_g: ratio of good trusted peers, t: Quorum size

Table 5.8: Experiments for System-key Maintenance

#	T	t	β	#	T	t	β
1.1	100	13	26	2.1	100	15	30
1.2	500	14	28	2.2	500	17	34
1.3	1000	14	28	2.3	1000	17	34
1.4	2000	14	28	2.4	2000	17	34

Legend: T: Number of trusted peers, t: quorum size, β: Size of update group

The URSA update and self initialisation strategy uses in the beginning an update group B, called proximity group with the simulation. Its size β must have at least the size t in order to enable recoveries by that group. Within the simulation, the update group size β is set to $\beta = 2t$ in order to achieve redundancy.

5.5.2.1 Parameters

In order to evaluate the scalability of the update and self initialisation scheme, different numbers of trusted peers T to be updated and recovered will be simulated. As the number of trusted peers is limited, trusted peer system size is between $T = 100$ and $T = 2000$ trusted peers.

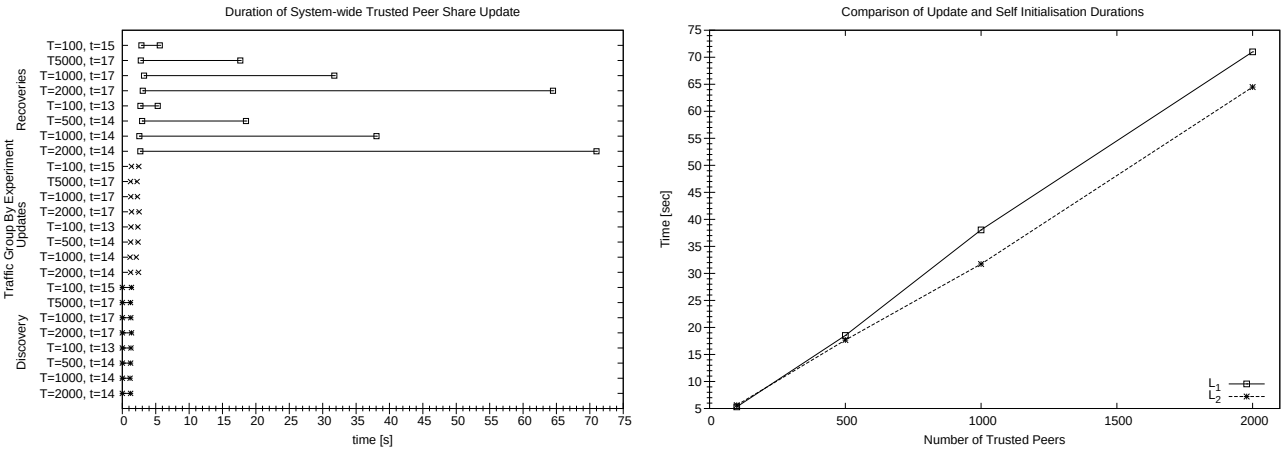
The experiments will be executed with different Trust Levels in order to observe the effect of a changing quorum size t in the created traffic. The computation of required quorum size depending on Trust Level L, number of trusted peer T, and ratio of good trusted peer p_g was presented in Section 5.2.1. As the required quorum size differs for one Trust Level depending on the number of trusted peers in the system T, the number of examined Trust Levels can be reduced. The resulting quorum sizes for 100, 500, 1000, and 2000 trusted peers is given in Table 5.7.

5.5.2.2 Experiments

The objective of the experiments is to assess the development of the created traffic. Therefore, four different parameter values are simulated for the main parameters system size T and quorum size t. This allows determining which kind the traffic develops for different parameter valued. Like for the token-based accounting simulations, a ratio of good peer $p_g = 0.5$ is assumed. Table 5.8 summarises the executed experiments.

5.5.3 Results

The update and self initialisation mechanism is a very deterministic process compared to the simulations of the token-based accounting scheme. The update and self initialisation simulation does not employ



(a) Durations by partial mechanism (b) Comparison of durations by system size

Figure 5.55: Durations for update and self initialisation

any probability distributions. Accordingly, the confidence intervals are very small and cannot be seen in the figures.[3]

Duration of Update and Self Initialisation Process

Before observing the traffic created, the time required for completely updating the trusted peer system is looked at. Figure 5.55 a shows the processes' duration.[4] The discovery of the initial update group requires between 1.16 seconds and 1.36 seconds. The difference is due to message transmission delays. The update phase required a bit less time and needed between 0.98 seconds and 1.18 seconds. the reason is that lookup messages might require several hops on the way towards the peer, where update requires direct communication. The recovery phase required 2.58 seconds for systems with $T = 100$ trusted peers and a quorum size of $t = 13$ and 68.3 seconds for a system size of $T = 2000$ and a quorum size of $t = 14$ trusted peers. The average time for a quorum size of $t = 17$ trusted peers is here a bit lower, which indicates that the quorum size has no influence on the duration of the recovery process. The reason is that the number of communication steps required within a recovery of one trusted peer is constant. Figure 5.55 b) shows the development of the recovery process durations more clearly. The time grows linearly. In the case of a Trust Level $L_1 = 99,99\%$ the best fit curve is $\tau_{USI} = 34.782T + 1911$ with a coefficient of determination $R^2 = 0.9989$; in the case of a Trust Level $L_2 = 99,999\%$ the best fit curve is $\tau_{USI} = 30.966T + 2004.1$ with a coefficient of determination $R^2 = 0.9989$.

The difference in the duration stems from the different sizes of the beginning start-up group. With a higher Trust Level, the update group in the beginning is larger; updating peers happens in parallel. Therefore, the update duration does not increase with the update group size. However, with a larger update group size less recoveries are required. Therefore, for $L_2 = 99.999\%$ the update duration is shorter.

Traffic Generated by Update and Self Initialisation

The traffic generated by the update and self initialisation process happens in a very short period of time, compared to the token-based accounting simulation. Also, the traffic will be distributed over all trusted peers. Therefore, showing the system-wide traffic of the complete process will not give deep

[3] The figures in this section have been formatted differently then the figures presenting the traffic of the token-based accounting scheme in order to make the recognition of the reference of a figure in the simulation easier.

[4] Note that the simulator did not simulate computation time. The durations all stem from message transfer.

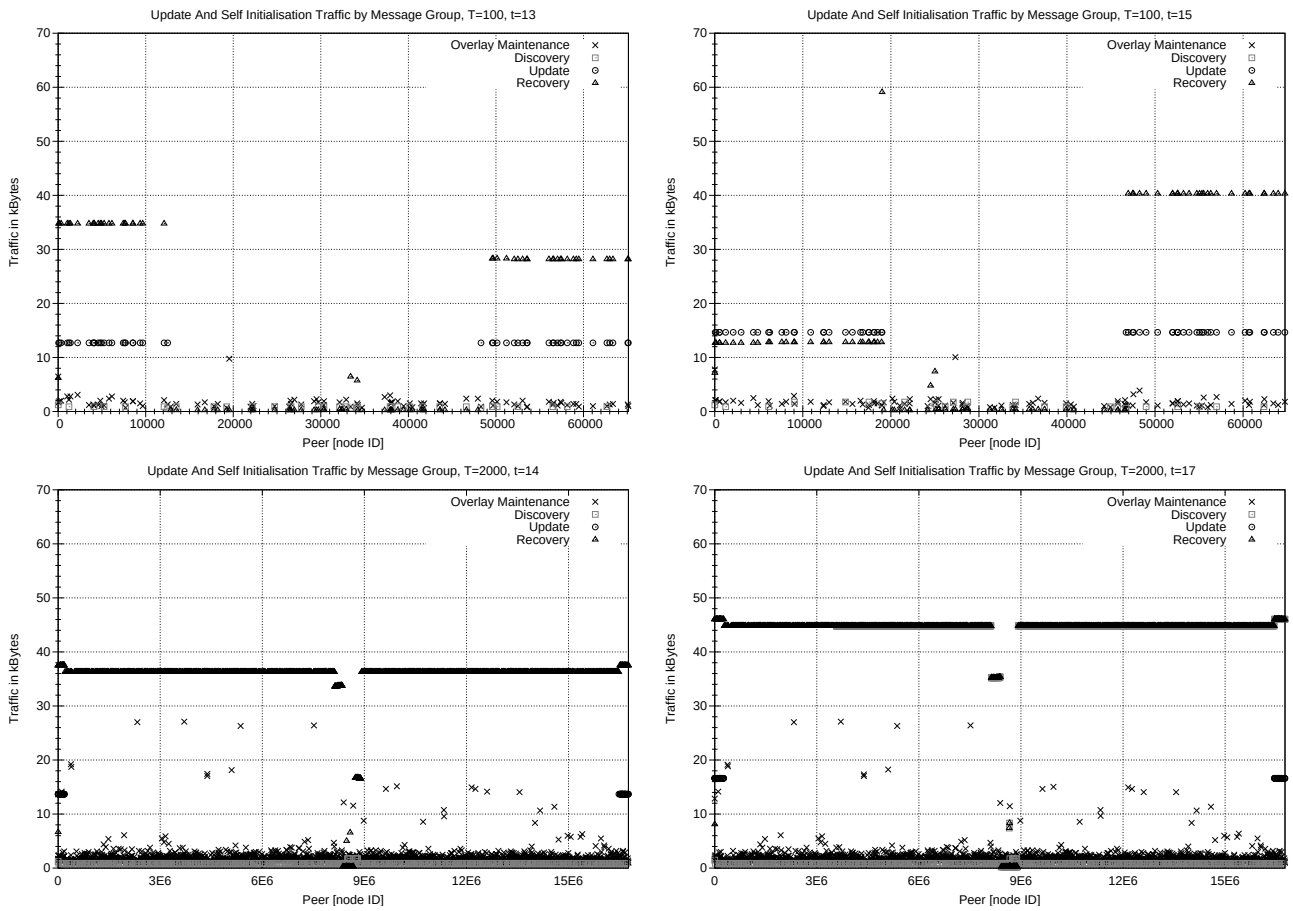


Figure 5.56: Update and self initialisation traffic by message type

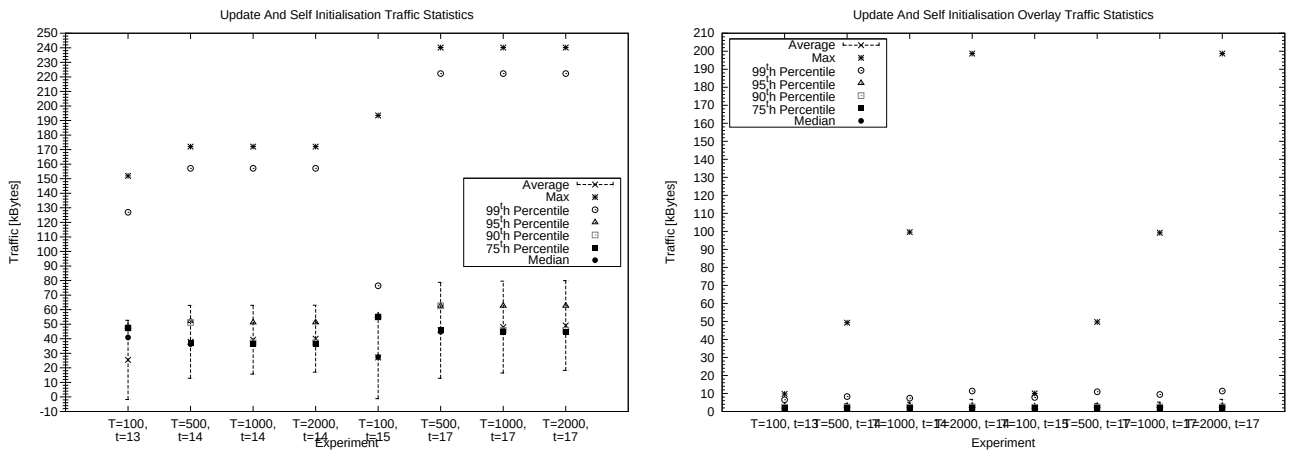
insights about the traffic load per trusted peer. Accordingly, the traffic will be observed on a per-peer basis.

Figure 5.56 shows the traffic generated by the update and self initialisation mechanism per peer for $T = 100$ and $T = 2000$ trusted peers and for Trust Levels $L_1 = 99.99\%$ and $L_2 = 99.999\%$ with the corresponding quorum sizes. The x-axis represents the peers' ID.[5]

It can be observed that the update group is symmetrically located around the peer ID 0. Especially in the graphs for large system sizes with $T = 2000$, the recovery traffic is almost equal for all peers. Only at the middle of the graph is the traffic distinctively lower, because here the two recovery fronts meet. Recovery traffic is the major source for traffic, although it is below 60 kBytes in total per peer. It can be seen that recovery traffic and overlay maintenance traffic have the lowest proportions of traffic load. Overall, a very even distribution of traffic load over the trusted peers is achieved by the update and self initialisation mechanism.

When observing the main source for traffic increase, it is obvious that the quorum size has an influence on it, however the system size influence seems minimal. In order to observe that more clearly, Figure 5.57 shows the descriptive statistics for the different experiments. Figure 5.57 a) shows the complete update and self initialisation traffic without the overlay traffic. Overlay traffic is depicted in Figure 5.57 b). Overall, the observations about the traffic distribution are confirmed. The average traffic of a trusted peer slowly increases over the experiment. For $T = 100$ and $t = 13$ average peer traffic is 25.46 kBytes for the complete process, with $T = 2000$ and $t = 14$ average peer traffic is 40.04 kBytes. For $T = 100$

[5] For $T = 2000$ the ID space was increased in order to enable complete message logging. Therefore, peer IDs are distributed from 0 to 2^{24}.



(a) Update and self initialisation traffic statistics

(b) Overlay traffic statistics

Figure 5.57: Traffic statistics for the update and self initialisation strategy

and $t = 15$ average peer traffic is 27.17 kBytes and for $T = 2000$ and $t = 17$ average peer traffic is 49.10 kBytes.

It can be seen that there are very few peers with distinctively higher loads. The maximum load for the complete update and self initialisation process is 240.13 kBytes, which happened in all experiments with $t = 17$. Assuming for trusted peers a DSL-connection with 128 kBit upload, a trusted peer would need 15.37 seconds to send this traffic. Accordingly, even the very few trusted peers with the maximum load do not get overloaded by the update and self initialisation process. The statistics show that there are only very few peers with a traffic load above 100 kBytes.

Observing the development of the maximum traffic, the best fit curve results in $v_{URSA} = 0.6374k^2 + 2.9283k + 6.1428$ kBytes with a coefficient of determination $R^2 = 1$.

It is interesting to see that the overlay also created a very similar traffic load. In each experiment, there was one peer that was distinctively higher loaded than the other peers. However, the highest loaded peers by the update and self initialisation process and the overlay are not the same. Accordingly, there is no relationship.

5.5.4 Summary

The simulations of the update and self initialisation mechanism, which was suggested as proactive mechanism for the URSA threshold scheme (LL00, LKZ+04), showed that updating a complete trusted peer system is practical in terms of traffic load per peer, as well as the time required to perform the update. Updating a system of 2000 peers required approximately 71 seconds. In terms of traffic, the highest loaded peers had to bear a load of approximately 240 kBytes. Maximal traffic growth is mainly to an increased threshold t of the threshold scheme. The best fit curve shows a small quadratic component. The average traffic load per peer is distinctively lower and varied in the experiments between 25.46 kBytes for a threshold $t = 13$ and a system size of $T = 100$, and 49.10 kBytes for a threshold $t = 17$ and a system size of $T = 2000$. Average traffic grows for smaller system sizes faster than for large system sizes. The increase between $T = 500$ and $T = 1000$ is approximately 2 kBytes and between $T = 1000$ and $T = 2000$ it is only approximately 1 kBytes; thus, its influence quartered.

Comparing the simulation results with the analytical results from Section 4.6, the simulations showed a slight increase in traffic. This is due to the increased details in simulations, like the considerations of peer discovery.

In summary, updating the shares of all trusted peers of the token-based accounting scheme is practical. It creates a surprisingly low load for the trusted peers. Such an update could be executed very frequently, for example once a day. Therefore, the secrecy of the private system key can be well protected.

5.6 Comparison with Karma

In Chapter 2, Section 2.3, it was concluded that Karma (VCS03) is closest to the goal of this dissertation. Karma allows querying a bank set for complete accounting information about all transactions a peer did system-wide. Furthermore, Karma allows cooperation control by introducing a virtual currency. Because other accounting mechanisms for p2p systems have less similarity in functionalities, comparisons with other systems are not appropriate.

This section will present a short comparison of the token-based accounting scheme with Karma. The trustworthiness as well as the traffic generated by the two accounting scheme is compared analytically.[6]

5.6.1 Trustworthiness Comparison

The comparison about the trustworthiness of token-based accounting scheme and Karma discusses how an adversary could enhance a peer's account balance.

Token-based Accounting Scheme

The token-based accounting relies on two mechanisms that ensure the scheme's trustworthiness; when tokens are issued they must be signed by a quorum of trusted peers. The secrecy of the required private key is guaranteed by using proactive secret sharing. Furthermore, information about all tokens issued to a peer are stored in an aggregation account, which is hosted at an account holder set.

If an adversary wants to increase the tokens available to a peer against the rules of the scheme, the adversary has to forge tokens and to put the information about these tokens into the corresponding aggregation account. Forging tokens requires either the knowledge of t shares of the scheme's private key the control of a complete quorum. The base scenario uses a quorum size $t = 17$. The account holder set hosting the aggregation account of a peer is an additional security mechanism. In order to manipulate an aggregation account, the account information at more than half of the account holders has to be manipulated. Also, the manipulation has to meet the specific requirements of the trustworthy token aggregation protocol, which is hard to fake. In the base scenario the account holder set size k is set to $k = 10$. Furthermore, the account holder set's location is concealed.

In summary, an adversary has to successfully attack more than 22 specific peers in order to defraud the system.

Karma

Karma's only mechanism for achieving trustworthiness is the bank set. In Karma the bank set size is denoted k. A bank set of a peer A is located at the leaf-set of the node closest in the nodeID-space to $HASH(nodeId(A))$ (see (VCS03)). The consistency mechanism of the bank set is based on majority, similar to the consistency mechanism of the token-based accounting scheme. Accordingly, an adversary has to successfully attack half of a bank set in order to defraud the system.

Conclusion

In order to achieve trustworthiness in Karma similar to the token-based accounting scheme Karma requires a larger bank set than the account holder set of the token-based accounting scheme. Assuming

[6] There is only one publication about Karma (VCS03) that however omits many details. A comparison using simulations is therefore not appropriate as too many assumptions would have to be made.

in the token-based accounting scheme the key share is stored sufficiently securely so that trusted peers cannot exchange their key share with other trusted peers, Karma would require a bank set size of at least $k = 44$ in order to achieve the same trustworthiness as the token-based accounting scheme.

5.6.2 Traffic Comparison

The comparison of traffic created for both accounting schemes is performed on an analytical basis, since there are no simulation results available for Karma. Further, insufficient information is available about the account holder set maintenance in Karma. Therefore, in this comparison, the traffic per transaction is being evaluated. For bank set maintenance, the token-based accounting maintenance traffic is assumed.

Token-based Accounting Scheme

In the base scenario of the token based accounting scheme with quorum size $t = 17$ and account holder set size $k = 10$ a traffic per transaction of 293.58 kBytes on average is created. Furthermore, for locating the account holder set, one lookup message is required, that travels at maximum $\log(N)+x_i$ hops, where x_i is the account offset. The amount of lookups required does not increase with account holder set size.

Karma

In order to analyse the traffic created per transaction by Karma, the transaction procedure described in (VCS03) is analysed, which is depicted in Figure 5.58 a). The messages of step 1 are signed by peer A, the remaining messages are unsigned.

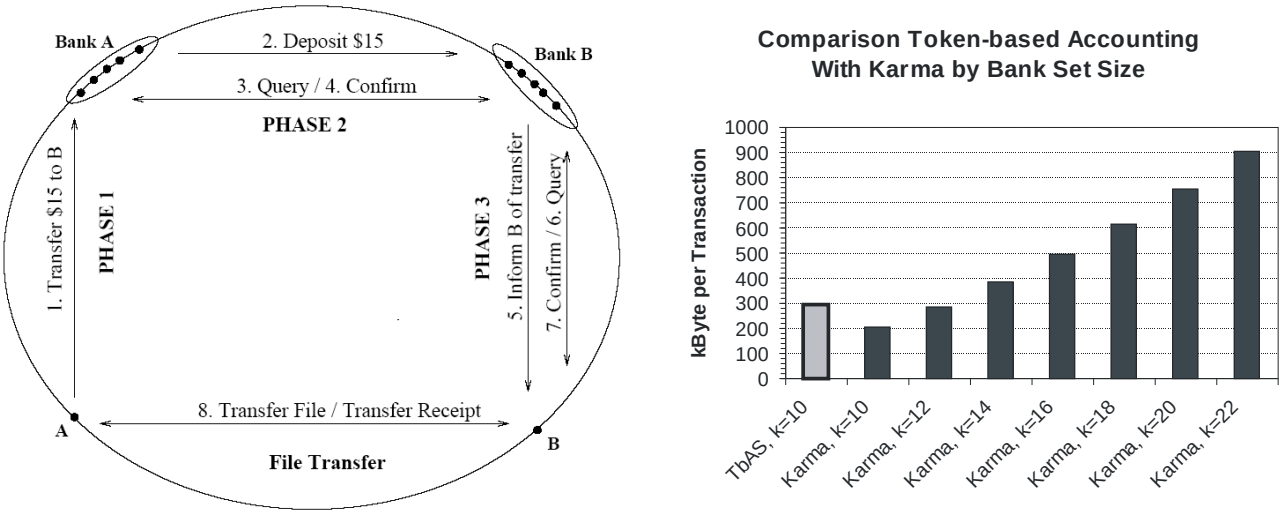
In order to do a fair comparison of the two accounting schemes, the message components used are similar to the token-based accounting scheme. The base message size for IP and TCP headers, sender and destination information, as well as further basic information is set to 805 bit. Additionally, unsigned messages require the peer identifications. For this, their public key is used. The key size is set to 1024 bit, similar to the token-based accounting scheme. Two keys are required, A's public key as well as B' public key. Furthermore Karma requires a serial number for each transaction. For this, a 160 bit number is assumed, as it is thought to be sufficient for avoiding message replay. For object id, SLA information 1600 bit are assumed, as it is for the token-based accounting scheme. Price information is an integer with 32 bit. Furthermore, in order to divide this information in a message, five delimiters of 16 bits each are required. Accordingly, an unsigned message has a size of 590.625 Bytes; a signed message has a size of 720.625 Bytes.

For the process depicted in Figure 5.58 a) 204.14 kBytes per transaction with a bank set $k = 10$ were used. This is almost 100 kBytes less than for the token-based accounting scheme. However, this does not take into account lookups. As each bank node in bank set A is required to find the bank set B, k lookups are required, each travels at maximum $\log(N)$ hops.

The comparison of the transaction traffic without lookups for different account holder set sizes in Karma as depicted in Figure 5.58 b). Increasing Karma's bank set size to 22 leads to 906.52 kBytes traffic per transaction; this is an increase by a factor of 3.09. Furthermore, the bank set maintenance traffic increases. It can be assumed that the maintenance traffic develops according to the best fit curve of token-based accounting scheme's maintenance traffic, which is $V_{AHS\,Maintenance} = 0.0225k^2 - 0.0674k + 0.1647$ MBytes. This results in 12.54 MBytes system-wide traffic per minute for $k = 22$, compared to 3.09 MBytes system-wide traffic per minute for $k = 10$. This is an increase by a factor of 4.06.

5.6.3 Summary

The analytical comparison of the token-based accounting scheme showed that the token-based accounting scheme offers a higher trustworthiness than Karma. In order to achieve a similar trustworthiness in



(a) Karma transaction procedure

(b) Comparison of traffic by bank set size

Figure 5.58: Karma transaction procedure (cf. (VCS03))

Karma, it requires a larger bank set size than the account holder set size in the token-based accounting scheme.

If the bank set size in Karma and the account holder set size in the token-based accounting scheme are both set to $k = 10$, Karma would create, per transaction, almost 100 kBytes less traffic. However, if the bank set size is set to 22 in order to achieve similar trustworthiness in both accounting schemes, Karma would require more than three times the traffic per transaction than the token-based accounting scheme. Furthermore, the bank set maintenance traffic would be approximately four times higher for $k = 22$ than for $k = 10$.

The improved efficiency of the token-based accounting scheme compared to Karma is achieved because the token-based accounting scheme uses primarily local accounts; remote accounts are only used as a mechanism for further enhancing the scheme's trustworthiness. This allows the number of replicas to remain small compared to accounting schemes that depend solely on remote accounts.

In summary, it can be concluded that for a similar trustworthiness the token-based accounting traffic creates significantly less traffic. This is true for traffic per transaction, as well as for static maintenance traffic.

5.7 Summary

In this chapter the token-based accounting scheme is evaluated using analytical methods and simulations. The evaluation criteria and methodologies are deduced from the objectives. This chapter's goal is to evaluate the efficiency of the token-based accounting scheme and to demonstrate its feasibility in a practical application scenario. Accordingly, the objectives are to evaluate the scheme's performance and costs.

In order to evaluate the systems trustworthiness, it is analysed which parameter setting of the token-based accounting scheme leads to which level of trustworthiness. The required quorum size is determined by concluding that the quorum selection process can be modelled by the hyper-geometric probability function. The Trust Level L describes the probability that the quorum is not completely compromised. The required quorum size is deduced for a set of different values of the influencing factors. Accordingly, for a Trust Level of 99% a quorum size of $t = 7$ is required, if at least 50% of the trusted

peers act according to the rules. For a Trust Level of 99.999% a quorum size of $t = 17$ is required, if at least 50% of the trusted peers act according to the rules.

Another core parameter influencing the token-based accounting scheme's trustworthiness is the account holder set size. Due to churn, account holder sets are reduced in size over time. Here, the mechanisms that repair the account holder sets' size was simulated for a set of eight different churn models and two different probabilities that peers handover aggregation accounts when they leave the system. The churn models simulated are two models deduced from measurements. Both models are modified with factors $d_{var} = \{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}\}$ in order to simulate higher churn and test the mechanisms in more difficult application scenarios. The results show that the account holder set size k should be set to values from $k = 6$ to $k = 10$ for the simulated churn models.

In the third section of this chapter, the mechanisms of the token-based accounting scheme are analysed in terms of their message complexity. The results show that the token-based accounting scheme's main mechanisms' traffic costs are independent of the p2p system's size. Both system-wide costs of the aggregation and payment processes depend mainly on the number of transactions performed; the quorum size t has further influence on the costs of aggregation. The account holder set size k has influence on both, aggregation and payment traffic. All these influences are expected to grow linearly with increased number of transactions or increased t or k. The account holder set maintenance mechanisms' traffic also grows with the p2p system's size, because each peer in the system is expected to own an aggregation account and account holder set maintenance is generated per account holder set. Part of these mechanisms have a message complexity $O(k^2)$; thus the account holder set size is expected to have a strong influence on the account holder set size.

In order to evaluate the token-based accounting scheme thoroughly, a simulation model is built using the PeerfactSim.KOM (Mul) simulator. PeerfactSim.KOM is specialised for simulating p2p systems and contains an abstract underlay model with message transfer delays for each overlay connection. In Section 4 the simulation model, simulation setup, experiments, and results of these simulations are described. The simulations study the effects of the quorum size t, the account holder set size k, and churn on the traffic created by the token-based accounting scheme. Overall, the generated traffic is moderate. The base scenario has maximal trustworthiness ($t = 17$) and sufficient reliability of the account holder sets for strong churn ($k = 10$). Here, the overall system-wide traffic generated by the accounting scheme is approximately 13.4 MBytes per minute for a system size of 1000 peers. That is approximately 30.30 kBytes per minute per peer. Splitting the traffic according to its source, a variable amount of traffic is created per transaction of approximately 293.58 kBytes. This traffic is distributed over the participating peers. A fixed amount of approximately 22.77 kBytes traffic is created per peer per minute for maintenance of the aggregation accounts. Observing traffic per minute, this represents the largest proportion of traffic. The quorum size increases the token aggregation traffic almost linearly, having only a very weak quadratic factor. For $t = 7$ aggregation traffic per transaction is on average 111.46 kBytes, for $t = 17$ it is 237.71 kBytes per transaction. The account holder set size influences the account holder set maintenance traffic. It increases for $k = 6$ from 3.35 kBytes per peer per minute to 10.74 kBytes for $k = 12$. This increase is a factor of 3.2 for a doubled account holder set size, which results mainly from the mechanisms with a message complexity if $O(k^2)$. These are in particular, the aggregation account locking mechanism and the aggregation account consistency mechanism. Churn also influences mainly the account holder set maintenance traffic. It increases from 7.70 kBytes per peer per minute for standard churn ($d_{var} = 1$) to 15.68 kBytes per peer per minute for a churn four times as intensive as standard churn ($d_{var} = 0.25$). System size has no influence on the traffic generated per peer or per transaction.

The peer's load is analysed by observing the peers' upload queue length. Assuming an upload bandwidth of 128 kBit per second in 97.85% of time peers have a load less than 1% of their upload capacity, and in 99.35% of time peers have a load less than 10% of their upload capacity. Trusted peers have to bear sometimes (0.65% of the time) a higher load of approximately 14 seconds queue length.

In conclusion, using simulations it is shown that the traffic overhead created by the token-based accounting scheme is overall low compared to typical p2p service traffic like file sharing of mp3-files or video-on-demand traffic. It is broadly applicable. The evaluation shows that within the account holder set maintenance mechanism there is still potential for optimisation, especially at the mechanisms having a message complexity of $O(k^2)$. For example, a locking mechanism better suited to p2p systems could be found. The mechanism used here was selected for the purpose of demonstrating the feasibility of the token-based accounting scheme. Researching an optimised locking mechanism is beyond the scope of this dissertation. Similarly, the correctness of the consistency mechanism was not evaluated because the research of consistency of replicas in p2p system also is beyond the scope of this thesis. Research performed in this field is, e.g., (MVHE04, MLTS08).

Section 5 observes the traffic generated by management of the scheme's private key also using simulations. A simulation model was implemented simulating the update and self initialisation strategy for updating the private key's shares within the trusted peers' overlay. For example for a system with $T = 2000$ trusted peers and a threshold $t = 17$ average traffic per peer of 49.1 kBytes is created. There are very few peers that have to bear a traffic load higher than 240.13 kBytes. The account holder set size has the strongest influence on the traffic generated per peer. The system size's influence decreases when the system size increases. An increase from $T = 1000$ to $T = 2000$ peers increases the traffic per peer by only 1 kBytes on average. Furthermore, the update and self initialisation phase is completed quickly. The complete update and self initialisation of $T = 2000$ trusted peers requires only approximately 68 seconds. Thus, in terms of traffic as well as in terms of time, updating the complete trusted peer system has such a low overhead, that there is no constraint on how often it is performed. It could be even performed once or twice per day, whatever the application developer believes is required in order to protect the scheme's private key's secrecy.

In Section 6, an analytical comparison of the token-based accounting scheme with Karma (VCS03) is performed. Karma is, in its functionality, the most similar accounting mechanism for p2p systems compared to the token-based accounting scheme. It is shown that if Karma is configured to a similar trustworthiness as the token-based accounting scheme, Karma creates 209% more traffic per transaction and 306% more maintenance traffic than the token-based accounting scheme. The improved efficiency of the token-based accounting scheme stems from the combination of using primarily local accounts for storing tokens and remote accounts for enhancing the scheme's trustworthiness. This keeps the account holder set small compared to mechanisms for relying solely on remote accounts.

In this chapter it is demonstrated that the token-based accounting scheme is applicable for practical p2p applications. Its novel architecture provides a unique combination of trustworthiness and small traffic overhead, while offering trustworthy accounting functionality. The scheme is secured by applying threshold cryptography in combination with aggregation accounts. Furthermore, the simulations showed the schemes robustness. Peers could perform transactions and swap tokens even under strong churn.

In conclusion, the token-based accounting scheme is suited for many application scenarios. It can be applied to p2p applications that provide high traffic volume services like video-on-demand as well as to p2p applications with low traffic volume services, because it creates a low amount of steady background traffic for account holder set maintenance.

6 Deployment Issues

After evaluating the token-based accounting scheme, issues about the deployment of the token-based accounting scheme in different application areas are presented in this chapter.

In the first section, it is discussed how the token-based accounting scheme could be bootstraped in a fully decentralised environment. The minimum requirements that have to be met for the start-up are derived, and it is shown how the scheme can be adapted to larger system sizes over time.

In the Introduction, two abstract application scenarios for accounting and cooperation control in p2p systems were discussed and the required functionality for the token-based accounting scheme was derived. In Section 2 of this chapter, it is demonstrated how the token-based accounting scheme could be applied in the first type of these application scenarios, commercial p2p applications, is discussed. Three alternatives are presented for using the token-based accounting scheme as basis for charging for commercial services.

The third section discusses the application of the token-based accounting scheme in the context of the second type of application scenarios, a community file sharing application. Using simulations, the effects that different aggregation functions have on communities are analysed. For peers, different behaviour types are assumed and the development of tokens available to peers File sharing behaviour and number of transactions performed are analysed under usage of aggregation functions that value uploads and downloads asymmetrically.

With this chapter, the research of fully decentralised accounting with intrinsic cooperation control in p2p systems closes the circle to the introduction. The two main application classes used there to derive the requirements for the token-based accounting scheme are also used in this chapter to discuss its deployment.

6.1 Bootstrapping the Token-Based Accounting-System

A p2p system requires a bootstrapping strategy, because many mechanisms in p2p systems assume the existence of a specific minimum number of peers. For the token-based accounting scheme, constraining factors for bootstrapping are the quorum size and account holder set size. Both mechanisms are built on cooperation of peers. Accordingly, for bootstrapping a p2p system that uses the token-based accounting scheme, three parts have to be considered. First, the overlay network for the token-based accounting scheme and the trusted peer overlay network need to be initialised. Then aggregation accounts have to be created for the first peers before they can receive the starting number of tokens.

6.1.1 Starting-up the Token-Based Accounting Scheme

The token-based accounting scheme applies two different overlay networks; the system overlay network is used by all peers, and the trusted peer overlay network is used for system key maintenance.

Minimum Required Number of Trusted Peers

The first step of system key maintenance is the distributed creation of the scheme's private key shares. Accordingly, the minimum number of peers for the trusted peer overlay network is determined by the scheme's private key creation process. The minimum number of trusted peers for start-up is determined

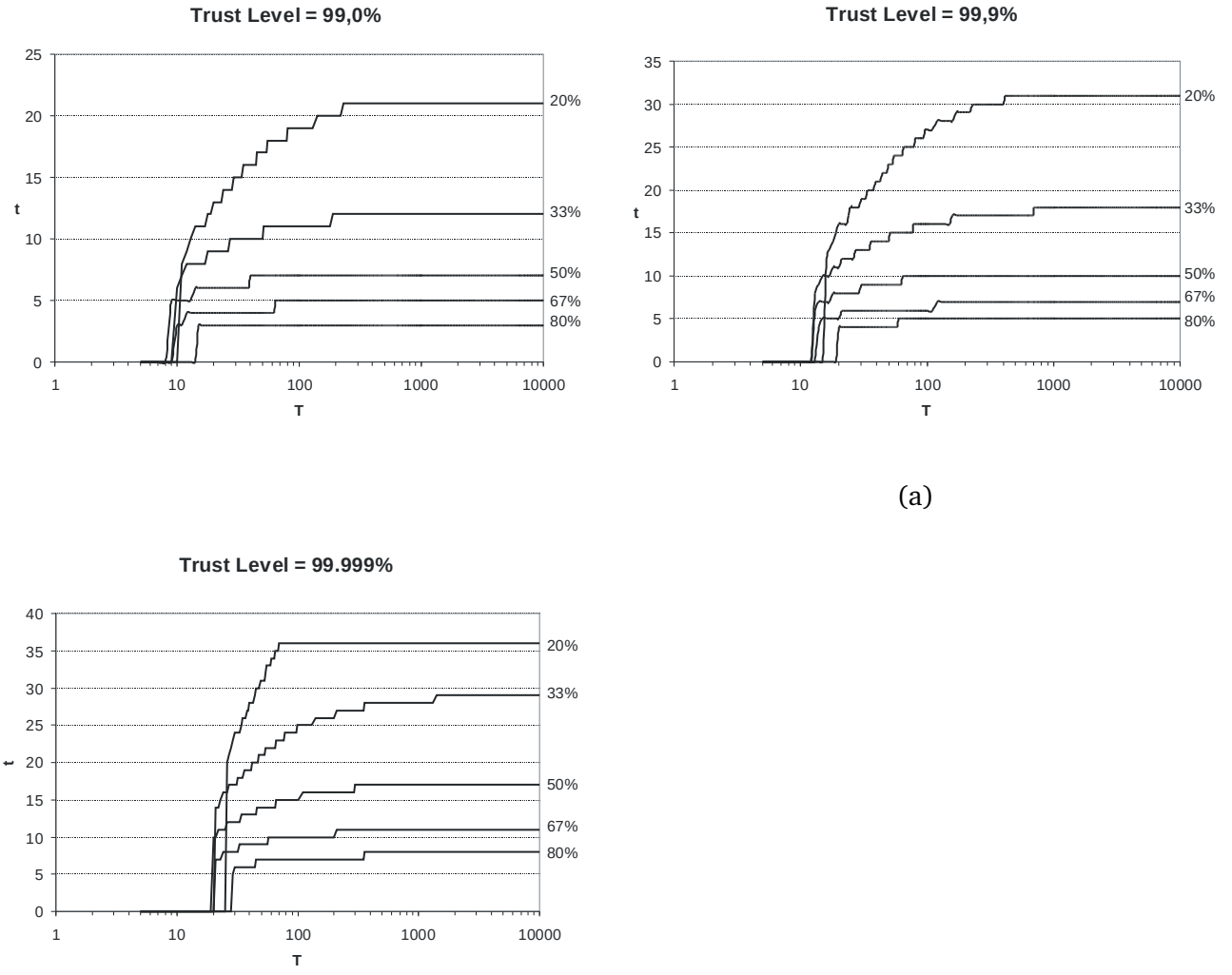


Figure 6.1: Required quorum size according to Equation 5.1

by the quorum size. The token aggregation process requires at least $t + 1$ trusted peers (the quorum and the token aggregation administrator). The quorum size depends on the number of trusted peers in the system, as well as the desired Trust Level. The Trust Level was analysed in Section 5.2.

For the start-up phase, it can be assumed that an application provider makes some machines available that may act as trusted peers. Accordingly, at the beginning, a very high proportion of honest trusted peers (p_g) can be assumed. Figure 6.1 depicts the required quorum size by number of trusted peers in the system for Trust Levels $L(T, t, p_g) = \{99\%, 99.9\%, 99.999\%\}$. Accordingly, a initial quorum size of 5 to 6 is sufficient.

If only this small numbers of trusted peers exists at start-up, there should be at least one trusted peer in the system whose trustworthiness is guaranteed. Then, the quorum selection process could be adapted to always use that peer while the number of trusted peers is small. This way, a trustworthy quorum can be guaranteed. Such a trusted peer could be operated by the application provider. If such a trusted peer is not provided, in order to ensure a trustworthy quorum selection process, the starting number of trusted peers should be at least 12 (see Figure 6.1).

The resulting quorum size can be increased over time in order to adapt to a changing ratio of honest trusted peers.

Minimum Required Normal Peers

The minimum number of peers in the system overlay is determined by the maximum of the initial account holder set size k and the initial number of trusted peers required, because both should not completely overlap in token aggregation processes. A peer should not be responsible for its own aggregation account. Therefore, the initial required number of peers is at least $t+k+1$. Furthermore, in order to use the account offset mechanism, a larger initial system size is desired. However, the account ship value can be set to a low value in the beginning (e.g., to $x = 3$) and can be increased with a growing system size. If an account holder set size of 6 is chosen due to churn, a peer becoming responsible for its own aggregation account must also be avoided. Therefore, assuming an account shift value of $x = 3$, and an account holder set size $k = 6$, at least 18 peers should exist.

In summary, for starting-up the token-based accounting scheme, there should be at least approximately 20 peers in the system overlay; at least 12 of them should be trusted peers if there is no special trustworthy peer provided by the application provider.

Start-up Client

In order to join a p2p overlay network, there must be at least one peer of the overlay network known by the joining peer. Apart from the mentioned minimum numbers of peers in the system overlay, and minimum of trusted peers, there exists no further constraint for starting up the token-based accounting scheme.

6.1.2 Scaling the Token-based Accounting Scheme

After start-up, when more peers join the system, its trustworthiness parameters must be adapted in order to guarantee trustworthiness.

The quorum size t can be increased by the trusted peers by running an update phase. Within the update phase, at least one polynomial with a higher degree is used for generating the update shares. This way the threshold of a threshold cryptography scheme can be increased. Unfortunately, decreasing the threshold again is impossible. If this is desired, a new key pair must be generated and the new shares must be distributed.

The account shift value is also increased by trusted peers when they perform a repositioning of an aggregation account.

Both mechanisms require that peers are able to approximate the actual system size. If the peers are organised in a structured p2p overlay network, this estimation can be performed by each peer individually. For example, in Chord a peer can estimate how dense the Chord ring is populated by observing its successors and fingers. From this, the peer can conclude to a system size. For the token-based accounting scheme, a coarse estimation of the system size is sufficient, because it is not used in any mechanism as parameter.

6.1.3 Summary

Apart from a specific minimum number of normal peers and minimum number of trusted peers, there are no additional constraints for starting up the token-based accounting scheme. Approximately 20 peers are required for start-up. With growing system size, the quorum size and the account shift value can be adapted.

The token-based accounting scheme could also be used with a central token aggregation server. Then one highly trusted server can be used for issuing tokens. This can be also an attractive alternative for starting up the token-based accounting scheme, because a smaller number of peers is required for start-up. This server could also be replaced anytime with a quorum of peers. In order to enable that, the method responsible for finding trusted peers must be adapted.

6.2 Deployment Scenarios for Commercial Applications

As described in Section 1.2, commercial application scenarios require charging and billing functionality in addition to accounting. Billing can be realised as an external application. In this section, there are three charging alternatives presented that can be added to our token-based accounting scheme. These alternatives are compared in terms of the transaction costs born by the peers.

6.2.1 Application Scenarios

This section covers three possible alternatives for charging within a p2p application where users pay actual money for receiving services. Thus, a p2p application providing the functionality as described in Section 1.2.1 is assumed. It is also assumed that each peer owns a private/public key pair which enables it to legally provide valid signatures. This means that before a service session starts, the peers agree on the service to be provided and a tariff for calculating the charge of the service.

In order to determine accounting information about the services and resources a peer provided, typically the foreign tokens a peer collected are used. Accounting information about the services and resources are available through copies of spent own tokens a peer stores. This information can be verified by the information stored in a peer's aggregation account, as well as by querying the transaction partners. This basic concept might be modified by in order to implement different accounting objectives.

6.2.1.1 Tokens as Receipts

Concept

The service requester (A) will send one or several tokens to the service provider (B) as receipt(s) for delivered service. B can use these tokens to demand payment from A via a prior agreed billing and payment system. Each peer can request any number of tokens using the token aggregation protocol. Tokens are not exchanged, only new ones are created.

Discussion

Here, tokens serve the same purpose as receipts created by transaction partners without having to be issued beforehand. Receipts not issued must remain non-forgeable and double spending has to be detectable. This, however, does not have to be system-wide, but can remain between the transaction partners. Both are easy to achieve through the use of signatures and unique receipt ID. Thus with the token-based accounting scheme, it seems unnecessary to issue receipts.

However, such issuing of receipts offers the possibility of decentralised control within p2p systems. For example, who is allowed to participate in the p2p system can be controlled. This can be used to exclude peers with a bad reputation. Furthermore, the number of tokens available to a peer can be limited. Thus, a peer can perform only a limited number of transactions between two token aggregations. This limits the danger of misuse of the reputation system, as seen on eBay; A person could be well behaved until he has a high reputation value; then suddenly he starts to defraud his customers by not sending the purchased goods. The person could continue at this for some time until it becomes clear that he is a fraud. The limitation of the number of tokens available to a peer is possible, because peers swap tokens only after a transaction is completed to mutual satisfaction. To further limit possibilities of fraud, for higher valued services, peers could agree on a higher number of tokens. The enhanced functionality described is especially wise for p2p business applications, as there is no central instance which users could contact in case of fraud (as there is in eBay).

In order to make fraud limitation effective for both the service requester and provider, the token-based accounting scheme has to be adapted so that both transaction partners must spend tokens for a

transaction. Both the requester when the service is received, and the provider when he receives the payment must spend tokens. Otherwise, only service requesters could be excluded from the system. Moreover, this allows efficient querying for accounting information, because each peer stores complete information about each of its transaction in a trustworthy manner.

It is apparent that this charging scheme also requires a fast payment scheme. Should the payment require, e.g., several days to arrive at the service provider (as in eBay), the p2p business application would be a lot less attractive.

6.2.2 Tokens as Micropayment

Concept

When using tokens as micropayments such as eCash (Sch98), each token symbolises a specific amount of money. Users use tokens to pay for receiving services.

Discussion

In comparison to existing micropayment schemes, tokens are not anonymous but can be modified to be (see Section 3.7 and Figure 3.3.2 a). When using tokens as micropayment, protection against forgery and double spending is highly important. The token-based accounting scheme provides these required functionalities. Without a central bank it is not trivial problem to solve whom users should pay in order to receive the tokens necessary for requesting services. A central bank to host the user's accounts and provide the token aggregation functionality would solve this issue. However, this compromises the p2p paradigm.

A solution without a central bank would require the cooperation of several banks with the (manufacturer of the) p2p system. A user would pay money to a participating bank, which would in return create a certificate that entitles the user to the receive a number of tokens. The peer (user) would present this certificate to a trusted peer for token aggregation in order to receive the tokens. It is important that the token-based accounting scheme ensures that certificates are redeemed only once. The peer's account holders can save this information in the peer's aggregation account or a callback function with the banks is possible.

An advantage of using tokens as a micropayment scheme is that peers could exchange foreign tokens received against new own tokens by using token aggregation instead of exchanging them at the bank. This reduces transaction cost.

6.2.3 Tokens as Bills of Exchange

Concept

Tokens can also be used as a bill of exchange. A bill of exchange is a written order in which one person pays another person a specific sum on a specific date. It can be enforced easily without being subjected to defences. In the past, the bill of exchange was a very important instrument for trading. Today, it used primarily in international trade. A token is worth the amount of money stated in it. Further information required for a bill of exchange (date of issue, drawee, recipient, and due date) must be contained in the token.

In comparison to micropayments, when using tokens as drafts there is no fixed value assigned to a token when issued. Like a normal draft, a token draft states the amount of money the drawee has to pay to the recipient. It also states the date when the drawee has to pay the money. The advantage of a draft is that it can be transferred by endorsement.

Discussion

This concept is similar to the first alternative (Section 6.2.1.1), however the legal consequences here are much more strict. Therefore, this concept has higher requirements on the peers' signatures, because they have the potential of being accepted internationally.

As an extended concept, a token used as a bill of exchange could be transferred by endorsement to another peer. The old recipient would add the new recipient under the token and sign it. However, now double usage of the token must be avoided (the old recipient could still claim the money from the drawee if he keeps a copy of the token). Therefore, the drawee must be informed about a transfer. If he is not available, the drawee's account holder set must store the information.

Tokens as bill of exchange also offer the opportunity for peers to charge up the mutually "drawn" tokens. This would save external transaction costs, which can be assumed to be higher than costs within the system.

Tokens as bill of exchange could be handled similar to tokens used as receipts, because each peer needs to get as many tokens as he requires for the services he requests (Section 6.2.1.1).

When applying this alternative, there is the danger of fraud by the transaction partner who has to deliver second (as explained in Section 6.2.1.1). Therefore, it is more secure to use several tokens in transactions if the service can be delivered in parts. To simplify the status quo of mutual debts, tokens with fixed amounts of money are preferable. Furthermore, the control mechanisms presented in Section 6.2.1.1 should also be applied here.

6.2.4 Assessment

In order to evaluate if a charging scheme can be applied in practise, the two most crucial criteria are scalability and security. The security of the presented alternatives for charging depends on the security of the token-based accounting scheme, which was analysed in Section 5.2. According to the requirements of the charging schemes, the trustworthiness parameters of the token-based accounting scheme are configured and the consequential costs are compared.

6.2.4.1 Configuration According to Security Considerations

In the following the charging alternatives are analysed individually in terms of the required quorum size and account holder set size for providing a sufficient secure token-based accounting scheme.

Tokens as Receipts

This alternative has the least security requirements compared to the other two alternatives. It is sufficient, if a defrauding peer must assume that double spending will be detected. Therefore, the account holder set can be kept small. An account holder set size $k = 6$ was selected for the traffic analysis. In order to calculate the quorum size, $p_g = 67\%$ is assumed and and a Trust Level of $L(T, t, p_g) = 99.9\%$ is required, which results in $t = 7$ for $T > 100$. If the total number of trusted peers is below 100 the required quorum size is $t < 7$.

Tokens as Micropayment

In order to enable enforceability of sale agreements, strong peer IDs are required. This scheme requires very tight security against forgery and double spending as this is equivalent to creating money. Furthermore, after foreign tokens have been swapped for new own tokens, it is much harder to prove that these tokens were created in the legal way. Therefore, known security mechanisms must be sufficient to ensure security of token creation. Some micropayment schemes introduce a lifetime of the virtual currency. For the token-based accounting scheme this is not required, because the lifetime restarts for tokens when they are swapped in the token aggregation process.

Assuming that 50% of the peers are bad and a Trust Level $L(T, t, p_g) = 99.999\%$, a quorum size $t = 17$ is required. In order to prevent double spending in scenarios with stronger churn, the account holder set size should be increased to $k = 10$.

In order to make forging the initial tokens created from a bank certificate impossible, these can contain information of this certificate, which can also be held by the account holders. For any tokens created hereafter, the other security mechanisms must be sufficient.

Tokens as Bills of Exchange

In this alternative, the transfer by endorsement is the most critical part, because different scenarios for cheating exist here. First, receiver B transfers a token to receiver C. Then B agrees with the drawee A to be paid 50% of the cost of the token. A would save 50% and B gains another 50% and C would not be able to collect the money from A. As time stamps can be easily forged, it is hard to decide which happened first, the token transfer or the payment of A to B. In order to prevent such fraud, strong peer IDs are required. Furthermore, the account holder set must always note the actual holder of a token and after each clearing of a token, remove it from its list. Accordingly, it is important that the account holder set is available and therefore its size needs to be increased to $k = 10$.

As token aggregation is primarily used for the limitation of fraud as in the "tokens as receipts" alternative, the quorum size is similarly configured to $t = 7$. In order for an effective limitation of fraud in this scheme, it is required that the service provider sends an own token to the service requester. These tokens are not allowed to be transferred as they are not bills of exchange.

6.2.4.2 Scalability

The overhead traffic for the three charging alternatives is analysed using the simulation results presented in the last chapter; for each charging scheme the different required configurations of the token-based accounting scheme are considered.

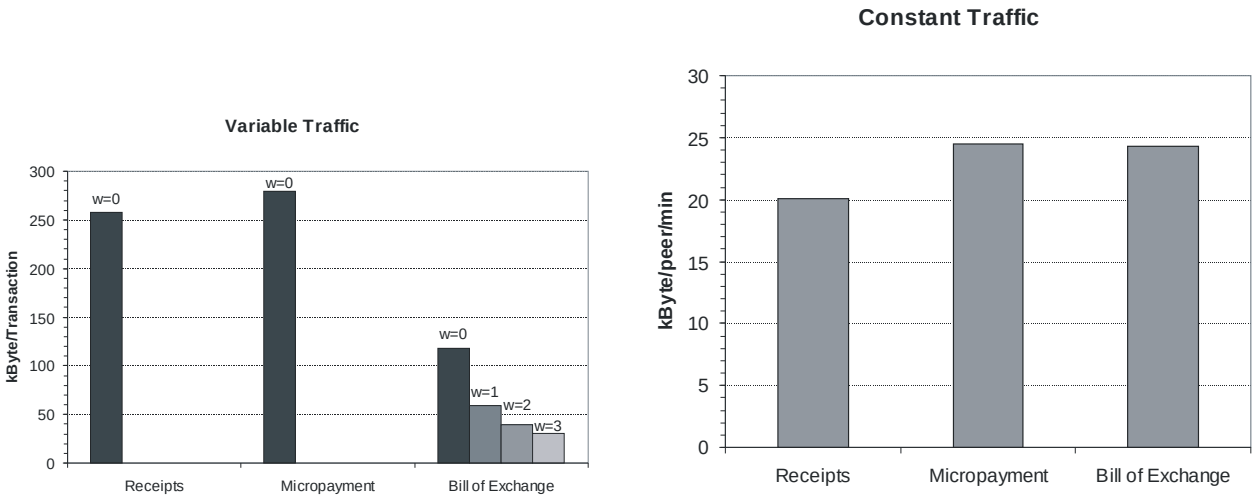
It is assumed that peers swap tokens in batches, and that on average after four transactions, a peer swaps its foreign tokens. This is approximately the same as in the simulation scenario; this is required to assume the same number of tokens swapped on average per token aggregation process, because this influences the traffic.

The variable and constant traffic of the token-based accounting scheme are compared. Variable traffic is the payment and token aggregation traffic. The remaining traffic classes (account holder set maintenance, Chord lookups, and Chord maintenance) represent the constant traffic.

Tokens as Receipts

The scalability of charging based on receipts is evaluated using the simulation results of the scenarios for $t = 7$, $k = 10$, $d_{var} = 1$ and $t = 17$, $k = 6$, $d_{var} = 1$, because the required scenario $t = 7$, $k = 6$, $d_{var} = 1$ has not been simulated. For payment traffic 10.13 kBytes per transaction is assumed, which was the result for the scenario $t = 17$, $k = 6$, $d_{var} = 1$. This can be applied because the quorum size does not influence the payment traffic. For token aggregation traffic, the ratio of token aggregation traffic of the two scenarios is calculated and then the traffic for $t = 7$, $k = 10$, $d_{var} = 1$ is multiplied with this ratio. This results in 474.33 kBytes traffic per token aggregation. In comparison to a file sharing scenario, double the number of token aggregation processes will have to be executed, because both the requester and receiver use tokens in these transactions and both have to swap them. Accordingly, the variable traffic results in 257.42 kBytes per transaction.

The remaining traffic classes (account holder set maintenance, Chord lookups, and Chord maintenance) represent the constant traffic; here the simulation results for $t = 17$, , $k = 6$, $d_{var} = 1$ are applied, as these are not influenced by the quorum size. This results in 20.05 kBytes per minute per peer traffic. The resulting traffic compared to the other charging alternatives is depicted in Figure 6.2.



(a) Charging using bills of exchange (b) Traffic comparison of the charging alternatives

Figure 6.2: Generated overhead traffic

Tokens as Micropayment

When using tokens as micropayment, the traffic created by the token-based accounting scheme is comparable to the traffic generated when tokens are used, as an incentive in a file sharing scenario (see (LDMS05)). However, the system parameters have to be adjusted according to the security requirements (see last section). Accordingly, for the results presented in Figure 6.2 b) a quorum size of $t = 17$ and an account holder set of $k = 10$ was assumed. The resulting variable traffic is 279.64 kBytes per transaction process and a constant traffic of 24.50 kBytes traffic per peer per minute.

Tokens as Bills of Exchange

The traffic overhead of this alternative is similar to the file sharing scenario of (LDMS05), however, the possibility of token transfer by endorsement has to be considered. Paying for a service with a token that the requester received as bill of exchange means that message sizes are larger, but fewer token aggregations are executed. The effect of this coherence is shown in Figure 6.2 a), where w is the average number of transfers by endorsement.

Using tokens as bills of exchange, only one token is required per transaction. Accordingly, the simulated payment traffic of 10.83 kBytes per transaction has to be reduced by approximately 5 token sizes. In the simulation, one foreign token has a size of 1096 Bytes. Assuming endorsements, tokens grow in size. Added to a token are a receiver ID, a signature, and further information. Here, it is assume with each endorsement, a token grows by 375 Bytes (two 1024 bit values for receiver ID and signature plus additional information). This also has to be taken into account for token aggregation.

Without endorsement, the variable traffic is approximately 117.63 kBytes per transaction. If each token is endorsed on average once ($w = 1$), the approximate traffic per transaction is reduced to 59.18 kBytes, because only half of the token aggregations processes are performed. For $w = 2$, the traffic per transaction is 39.82 kBytes, for $w = 3$, the traffic per transaction is 30.23 kBytes.

The constant traffic is 24.31 kBytes per minute per peer.

6.2.5 Summary

This section presented three alternatives for charging based on the token-based accounting scheme. In general, the use of tokens has its advantages compared to using simple receipts. Particularly, the number of tokens available to a peer can be limited. This can be used either as a mechanism to resolve market failure, or as a mechanism for limiting fraud possibilities, as the number of transactions which a peer may execute can be limited. In a p2p environment, this is especially important as the transaction partners are widely anonymous and therefore mechanisms which build trust are required. Identity management plays an important role as it is required in order to be able to identify defrauding peers clearly. The additional functionality and controlling the token-based accounting scheme offers results in the generation of higher traffic overhead. For using simple transaction receipts as a variable traffic of approximately 9 kBytes per transaction can be assessed; here no constant traffic is created.

The advantage of charging based on micropayments or bills of exchange is the possibility for peers to charge up mutual debts and by doing so to save on banking fees. Especially when using tokens as micropayment, peers receive their payment immediately. This means that customers can retrieve the requested service immediately, and do not have to wait for a bank confirmation.

Security aspects become very important in p2p systems as soon as it involves real money. The charging scheme presented using tokens as micropayments can be considered secure. However, there are some limits which might influence the trust a user has in a micropayment scheme based on token-based accounting. For example, tokens issued due to deposit of money in a users account contain a certificate signed by a bank. However, after token aggregation, a new token cannot contain such a certificate anymore. Therefore, it is questionable if users of banks would accept a micropayment scheme which relies on a decentralised mechanism without a trusted third party.

6.3 Economics of Virtual Currency Based Incentive Systems

Community applications built on p2p systems create the largest amount of traffic in the Internet (Has05). These are mainly file sharing applications. Furthermore, a decentralised video on demand, e.g., a p2p-based YouTube can be envisaged. Here, it is important that the incentives for collaboration are configured correctly. Otherwise the applications performance is reduced.

In (KSTT04) it was shown that in p2p file sharing, some free-riding can be tolerated in the social optimum, which is not addressed by the proposed incentive mechanisms. A weaker incentive mechanism would be appropriate where peers would not be required to provide as many resources as they consume. However, it is a matter of fairness that every peer should contribute some resources to the system and complete free-riding should not permitted.

In this section, the effects of different exchange ratios for new tokens against foreign tokens is researched. The means for using different exchange ratios is the aggregation function within the token aggregation process. However, if for each swapped foreign token more than one new token is created, with every service provided more virtual currency is created. This will lead to inflation and could finally result in a collapse of the incentive system when tokens are not scarce anymore.

In each community there exist altruistic peers that collaborate with other peers regardless of the existence of an incentive system. The basic concept to overcome the inflation effect is to capitalise the altruistic peer behaviour. Altruistic peers would accumulate the additional created tokens and consequently the system may stay in a steady state. This concept is researched using simulations.

6.3.1 Simulation Model

The goal of the simulation model is to determine if, in a p2p file sharing scenario, asymmetric incentives make sense. Thus, an asymmetric incentive system must fulfil two criteria. On the one hand, no inflation must occur when asymmetric incentives are used in order to ensure that the incentive scheme is stable over time and its effects are not diminished. On the other hand, it must be shown that the asymmetric incentive mechanism is functioning, i.e., it has an affect on user behaviour. To implement asymmetric incentives into a p2p system, we use the token-based accounting scheme by choosing an aggregation function different than $m = n$, where M denotes the number of new tokens $\{T_1, ..., T_m\}$created due to aggregating n foreign tokens $\{F_1, ..., F_n\}$. To value upload twice as much as a download, the aggregation function would be set to $m = 2n$.

In order to assess if inflation occurs in the system, we have to define a condition to determine which tokens are still used for trading in the system and which not. That is, we have to determine which tokens belong to the cash flow in the system. In order to assess whether the incentive system is functional, it has to be determined if peers' file sharing behaviour is improved in comparison to peers' file sharing behaviour without incentive system. In order to accomplish that, peer behaviour models for strategic peers, normal peers, and for altruistic peers are defined.

6.3.1.1 Peer Behaviour Models

In order to model realistic peer behaviour, three different file sharing behaviour models are used. Each peer belongs to one of the following peer classes:

- Strategic peers that do not share any files voluntarily. They represent free-riders.
- Normal peers that merely share a small number of files. They correspond to peers that download files and share them afterwards for some time.
- Altruistic peers that share all files they have.

In the model, it is assumed that peers upload all files that are requested from them. Therefore in the model, the goal of an incentive system is to motivate normal peers and strategic peers to share more files.

6.3.1.2 Inflation Detection

In order to determine whether inflation occurs in the system, the number of tokens available to the peers targeted by incentive system must be calculated. If this number increases over time then inflation of tokens occurs. Typically, the peers targeted by the incentive systems are the strategic peers and the normal peers. Altruistic peers share their files independently of the presence of an incentive system. Therefore, altruistic peers are expected to accumulate tokens. These accumulated tokens are not available to the system anymore. In order to decide which tokens are still available to the system, it is defined that the tokens available to the systems are represented by the own tokens in the system. Furthermore, it is defined that the tokens not available to the system are represented by the foreign tokens in the system. Thus in the simulation, peers swap received foreign tokens immediately back to own tokens, if they plan to spend them again. Therefore, for every behaviour model, a different token aggregation policy must be defined:

- Strategic peers exchange all received foreign tokens immediately back to own tokens.
- Normal peers aim at having always a specific number of own tokens available.

- Altruistic peers also want to download from the p2p system. Therefore, they also try to have a specific number of own tokens available.

By applying these rules for file sharing and inflation detection, it can be argued that strategic peers do not accumulate any tokens. They merely spend tokens for downloading files. However, they only receive tokens when they share files (what they normally not do). On the other hand, altruistic peers will accumulate tokens, while normal peers always share a small number of files. The purpose of the incentive system is to motivate all peers to share more files. Therefore, normal peers could also accumulate tokens. However, that would mean the incentive system is not working for the normal peers anymore. Accordingly, the number of foreign tokens normal peers hold is a measure for the degree of inflation in the p2p system.

Obviously, a metric for the efficiency of the incentive mechanism is needed. The purpose of the incentive system is to motivate peers to share more files. Accordingly, the number of additional files normal peers share and the number of files foreign peers share is the measure of the efficiency of the incentive mechanism.

In order to evaluate the described peer behaviour model, a simulator was used that will be now described in detail.

6.3.2 Simulator

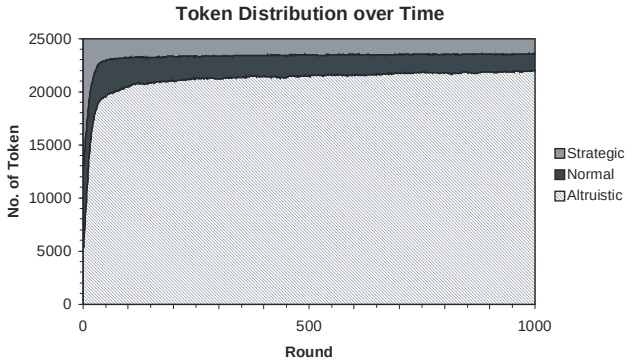
The simulator implemented for this evaluation is round-based.

At the beginning of every round, each peer has a specific number of own tokens T and foreign tokens F and shares a specific number of files ϑ. In addition, each peer belongs to one of the three behaviour classes. The goal of the simulation is to assess for which percentages of altruistic and strategic peers asymmetric incentives can be applied. Accordingly, the behaviour class is invariable for each peer.

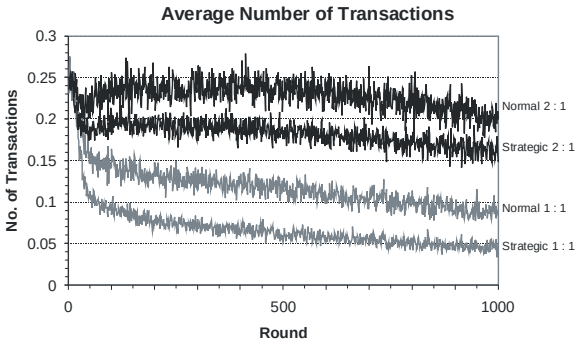
During each simulation round for each peer, it is determined whether it wants to download a file based on a predefined download probability. If so, it is determined which file the peer wants to download, as well as the provider peer. Each file has a specific price s in tokens. In this simulation, the focus is on the inflation of the system, not on file distribution. It is assumed that files are evenly distributed among the behaviour classes according to their popularity, and that the interest in these files is also evenly distributed among the behaviour classes. Therefore, it is concluded that file popularity does not influence the inflation effect. Thus, file popularity is not modelled.

A download takes place if the requester peer has enough own token to pay the provider. Then the provider receives a number of foreign tokens that is equal to the file price s. The same number of own tokens is subtracted from the customer's own tokens. If the customer does not have enough own tokens to afford the download, the peer shares an additional number of files ϑ' in order to receive more upload requests. In the next round, the peer will try again to download the file. If it still does not have enough own tokens, it will again share some more files. This continues until the download request is successful. At the end of each round, peers' foreign tokens are exchanged against own tokens according to the peer's behaviour class' aggregation policy. In addition, the number of shared files is adjusted. If a peer downloaded a file successfully, the file sharing policy according to the behaviour model is applied. Otherwise, the peer will share additional files, as described before.

Finally, for each peer class, the number of tokens and shared files is calculated for evaluation purposes. It is assumed that peers show the same up-time behaviour in all behaviour classes. Thus, on average the percentages of peers of the behaviour classes in the system are invariable. Accordingly, peer up-time will not have an influence on the inflation effect. Therefore, peer up-time is not modelled in the simulation.



(a) Number of own token by peer class



(b) Number of transactions by peer class

Figure 6.3: Simulation results for symmetric aggregation function

6.3.3 Simulation Scenarios

In every simulated scenario, peers start with 25 own tokens and no foreign tokens. Further, each peer starts with a certain number of own files; this number is uniformly distributed between 5 and 100. The files have a normal distributed price s with a mean of 5 and a standard deviation of 3. Altruistic peers share all of their files. Normal peers share a random number of files, which has a Normal distribution with a mean of 10 and a standard deviation of 5. Peers want to download a single random file in a simulation round with a probability of 25%. Altruistic and normal peers exchange 15 foreign tokens to own tokens as soon as their balance of own tokens falls below 20.

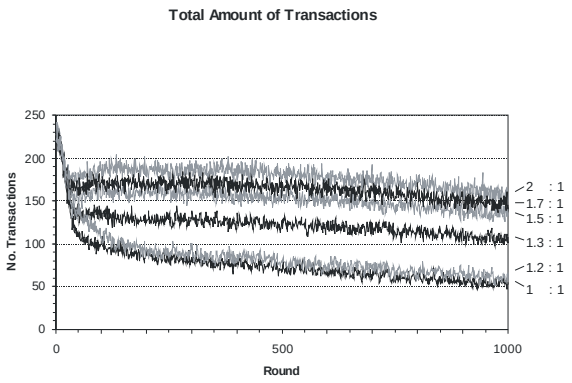
For the presented results, systems with 1000 peers have been simulated with varying ratios of altruistic, normal, and strategic peers. In empirically observed file sharing systems, there are approximately 10% to 20% altruistic peers according to (AH00, HLD+05). Strategically acting peers form the majority. In the results presented below the simulated systems had 20% of altruistic peers, 30% normal peers, and 60 % strategic peers.

6.3.4 Simulation Results

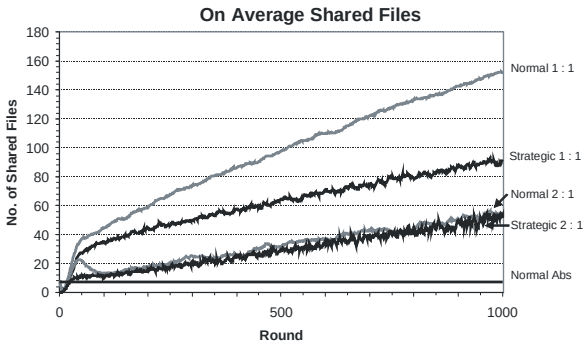
Figure 6.3 shows the simulation results for a symmetric aggregation function. As expected, the number of available tokens decreases sharply at the beginning. This is a strong deflation. After approximately 200 simulation rounds the development flattens and the number of tokens held by the altruistic peers increases only slowly. The number of transactions performed by peer class is shown in Figure 6.3 b).

Compared to the symmetric case, with an aggregation function $m = 2n$ strategic peers perform approximately three times as many transactions, normal peers a bit more than twice as many transactions. This observation is presented in more detail in Figure 6.4 a). Here, the sum of number of transactions for normal peers and strategic peers is depicted. An aggregation function $m = 1.2n$ does not increase the distribution speed in the file sharing scenario significantly. An aggregation function $m = 1.3n$ approximately doubles the performed transactions, which can be interpreted as a doubled distribution speed in file sharing applications. For $m = 1.5n$ the distribution speed almost triples compared to a symmetric aggregation function. For stronger asymmetric aggregation functions, the number of transactions still increases, but not as strongly. In conclusion, in terms of distribution speed, the optimum aggregation ratio is between $m = 1.3n$ and $m = 1.7n$.

Figure 6.4 b) shows the number of shared files by normal peers and strategic peers for $m = n$ and $m = 2n$. In the symmetric case, peers share many more files than in the asymmetric case. This is because

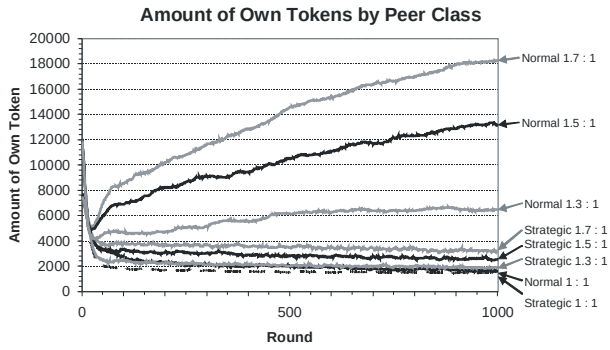


(a) Total amount of transactions by aggregation ratio

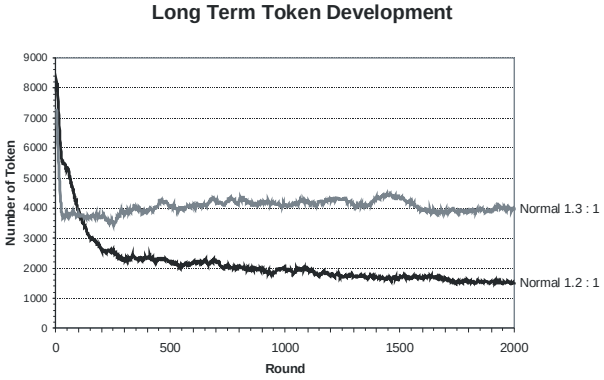


(b) Shared files by peer class

Figure 6.4: Number of transactions and file sharing behaviour



(a) Development of own tokens by aggregation ratio



(b) Long term development of own tokens

Figure 6.5: Development of own tokens

peers have very few tokens in this scenario. Obviously, the average number of shared files cannot be applied as a metric for fast distribution of files in the system.

Figure 6.5 a) depicts the development of own tokens over time. For all aggregation ratios the number of available tokens drops at first. After approximately 50 rounds, the number of tokens either starts to stabilise or to increase again. For aggregation ratios $m = 1.7n$ and $m = 1.5n$ inflation can be observed for normal peers. Accordingly, the incentive system will loose its impact on the behaviour of these peers eventually. For $m = 1.3n$, the number of own tokens seems to stabilise, which would result in a sustained incentive system. This is observed in more detail in Figure 6.5 b). The long term development shows that for the scenario with $m = 1.3n$, there is still deflation existing. However, with $m = 1.3n$ the number of available tokens is stable.

6.3.5 Summary

p2p systems are based on the assumption that participating peers share their own resources with other peers. However, file sharing applications have shown that only a minority of peers share their resources voluntarily. This reduces the system performance considerably, and the system operates below the social optimum (KSTT04). Therefore, incentive schemes are needed. They motivate or even force users to share more resources. However, at the social optimum, free-riding can be tolerated to a specific degree. Therefore, an incentive scheme does not need to be strict, so that not all peers need to provide at least as many resources as they consume. However, as a matter of fairness, every peer should still contribute some amount of resources to the system and complete free-riding should not be permitted. Further, today's Internet connections are in their majority asymmetric. Thus, the rationale of introducing asymmetric incentives to allow a specific degree of free-riding and to value an upload higher than a download is obvious. However, inflation of the virtual currency used to introduce incentives might occur. Due to the p2p paradigm, there is no central broker in the system that could control the virtual currency supply. Altruistic peers that accumulate the surplus currency counteract the inflation effect.

In this section, it was shown how to configure asymmetric incentives in order to compensate the inflation effect. The key finding of the conducted simulation is that the inflation effect for asymmetric incentives with ratios of $m = 2n$ and higher can not be compensated by realistic proportions of altruistic peers. However, it was shown for asymmetric incentive ratios $m = 1.3n$ that the inflation effect is compensated, and the system evolves to a steady state. This is true for all different percentages of altruistic peers we simulated. Thus, altruistic peers seem to offer a natural way to cope with inflation. With different percentages of altruistic peers (and incentive ratios below $m = 2n$), the system evolves at a steady state because the resulting sharing behaviour of the strategic peers differs. Further, it was shown that even with higher asymmetric incentive ratios than $m = 2n$, peers show considerably improved file sharing behaviour, which fulfils the main goal of an incentive system. However, it is possible that with such asymmetry the inflation saturates the system and the incentive system looses its impact.

Also, this analysis shows that the aggregation function can be used to coordinate the amount of resources available in a p2p systems, because an economy around tokens can be established. This can be also used to control the incentives for resource provisioning.

6.4 Summary

In this chapter issues arising from deploying the token-based accounting scheme in a real-world scenario were discussed.

In the first section, it was presented how the token-based accounting scheme can be started-up. There are a specific number of peers and trusted peers required if all mechanisms ensuring the scheme's trustworthiness are to be applied from start-up. Then, approximately 20 peers are required, of which at least 5 peers have to be trusted peers, because this is the minimum required quorum size. This assumes that an application provider provides some trustworthy peers. If this is not the case, at least 12 trusted peers are required. After start-up, the token-based accounting scheme can be adapted easily to larger system sizes. Using the key maintenance protocols, the threshold of the applied cryptography scheme can be increased. Also, account holder set size and the account shift value can be easily adapted by the peers, according to estimations of the system size.

The second section is about how the token-based accounting scheme can be applied in commercial p2p application scenarios. The requirements for the application of token-based accounting scheme in commercial scenarios has been discussed in Chapter 1. The second section presents three different alternatives for charging based on the token-based accounting scheme: Using tokens as transaction receipts, using tokens as micropayments, and using tokens as bill of exchange. Using tokens as receipts requires that both transaction parties have a receipt of the transaction. Therefore, both parties have to

spend tokens in a transaction. Token aggregation enables additional control that pure receipts cannot offer. Using tokens as micropayments enables banks to be included with the token-based accounting scheme. Each token has a fixed value in a real world currency. A peer has to deposit money in a bank account, and receives in return tokens that are created using the token aggregation process. A bill of exchange is a traditional trade concept. Here tokens, have a flexible value and can be transferred by endorsement to other peers. This saves costs. According to the security requirements of the alternatives, their traffic costs were analysed.

The third section discussed the application of the token-based accounting scheme for p2p community applications. Here, it was researched which effects different aggregation functions have on file sharing communities with different classes of peers. Systems with three classes of peers were simulated – normal peers, strategic peers and altruistic peers. It was shown that due to the existence of altruistic peers, a symmetric aggregation ratio for tokens leads to a strongly decreased system performance. Altruistic peers collect more tokens than they spend. Therefore, deflation occurs and the other peers do not own enough tokens for buying services. This can be avoided by choosing an asymmetric aggregation function with a ratio $m = 1.3n$. For higher asymmetry, the distribution speed in the system can further be improved, however then inflation exists and finally the incentive system will loose its impact when the system becomes saturated. This analysis shows that the aggregation function can be used to establish an economy with tokens within a p2p system; the economy can be applied to control the incentives for resource provisioning.

In summary, this chapter concluded the research of the token-based accounting scheme by demonstrating its application in the case studies that were used in the beginning to deduce the requirements for the token-based accounting scheme.

7 Summary, Conclusion, and Outlook

This chapter concludes this dissertation. First, the content and main findings of each chapter are briefly summarised. The second section summarises the contributions of the token-based accounting scheme to the domain and draws conclusions. The last section gives an outlook on new research challenges opened up by the work performed in this thesis.

7.1 Summary

Peer-to-peer (p2p) systems are still lacking mechanisms for *trusted accounting* and *cooperation control*; however they are two core mechanisms required for a wide range of applications. Commercial applications require accounting functionality, and community applications require cooperation control, which also requires accounting in order to attribute cooperation to users.

This dissertation demonstrates the *feasibility of fully decentralised trusted accounting with intrinsic cooperation control*.

Chapter 1 "Introduction" shows that accounting as well as cooperation control are required as core mechanisms for a wide range of p2p applications. Commercial applications need an accounting functionality, and community applications require cooperation control, which also relies on accounting in order to attribute cooperation to users. Accordingly, the goal was identified to demonstrate the feasibility of fully decentralised trusted accounting with intrinsic cooperation control.

Chapter 2 "Related Work" is structured into three sections. The first section gives relevant definitions in the context of the dissertation. In the second section, the related work in the area of accounting in computer networking (apart from p2p systems) is reviewed. Here, accounting functionality is performed at a central trusted server, which is not compliant with p2p. In the third section, accounting mechanisms specifically in p2p systems are reviewed in detail. A classification for p2p accounting mechanisms is presented, and it is concluded that so far no trustworthy accounting mechanism for p2p systems existed. Further, there has been no accounting scheme that stores accounting information using local accounts in a trusted way. However, this would be required for $O(1)$ messages during transactions. Furthermore, such a solution would not rely on trusted peers during a transaction for correct accounting information. In the fourth section, an in-depth analysis of security mechanisms suited for decentralised systems is performed. Threshold cryptography was selected as the class of security mechanisms that fulfils the largest set of requirements identified for applications within p2p systems. A thorough analysis of these cryptography schemes revealed that most are limited to scenarios with a small number of key shares. However, two threshold cryptography schemes are identified that fulfil the requirements for an application in large p2p systems. These can be applied for building a distributed basis of trust in p2p system.

Chapter 3 "Token-based Accounting Principles and Architecture" introduces the design of framework, system architecture, and protocols of the token-based accounting scheme step by step. For each step, the design alternatives are analysed and a design decision is concluded. The result is the token-based accounting scheme, which used tokens as a combination of permission objects for access to accounted resources and services, and as transaction receipts. Tokens are issued to a specific peer. Peers spend tokens in transactions with other peers. When a peer provides accounted resources or services, it collects tokens issued to the receiver. Peers swap these collected tokens against new own tokens in the fully decentralised token aggregation process. The token-based accounting scheme consists of the four closely interlinked building blocks *token structure*, *transaction protocols*, *token aggregation protocol*, and

detection of double spending. For each building block, the basic protocols are presented. The token structure ensures authentication, integrity, and non-repudiation of accounting information. The trustworthy transaction protocol removes the benefits of defrauding the transaction partner. Token aggregation ensures that tokens are created only according to the token-based accounting scheme's rules and cannot be forged. No central trusted entity is required, because a distributed basis of trust is used applying a threshold cryptography scheme, which has been identified in Chapter 2. Further, the token aggregation process introduces the aggregation function that determines the number of new tokens a peer receives. With this function, another degree of freedom is introduced in the token-based accounting scheme that allows implementing different rules and policies in the p2p system. Detection of double spending is performed in a fully decentralised and efficient way by introducing aggregation accounts that are located at a set of third party peers - the account holder set. The combination of the four building blocks prevents cheating and collusion of peers.

Chapter 4 "Relevant Details about the Protocols" focuses on several parts of the token-based accounting scheme and presents details about the developed mechanisms. The first section described the identification scheme for peers, which ensures that peers can be clearly and permanently identified. This prevents Sybil and whitewashing attacks. In the second section the location, creation, access, and maintenance of account holder sets is discussed. For trustworthy access to aggregation accounts hosted at the account holder sets, a novel routing mechanism was developed that achieves receiver anonymity in the presence of sender identification. Existing mechanisms for anonymous communication assume that sender and receiver of a message want to stay anonymous and break if any communication party must be identifiable. Clear sender identification is required as it impedes attempts at cheating, collusion, and malicious behaviour, because such attempts can be traced back to its source. The novel mechanism can be applied on any structured p2p overlay network building a ring topology. For the aggregation accounts, a mechanism for consistent long-term remote storage of dynamic data in DHT-based p2p overlays under churn was developed. The mechanism consists of a set of protocols that detect the actual size and location of an account holder set, that lock an account holder set so maintenance actions can be performed without creating inconsistencies, that establishes consistency across account replicas, that move an account to a new position, and that allows peers to perform a graceful account handover before they log off. In the third section, the selection and organisation of trusted peers in a separate overlay was presented. The fourth section extends the token aggregation protocol with mechanisms enhancing its security. A mechanism was developed that assures rule compliant execution of peer tasks in the presence of incentives for fraudulent behaviour. The mechanism implements a random selection of the aggregation administering peer as well as the quorum. It permits the verification of the randomness of the selection of the peer by any peer, although the result of the selection process cannot be determined in advance. Therefore, which peers participate in trustworthy actions cannot be influenced. Cheating and collusion of the participating peers is impeded, and an adversary cannot take control of the participating peers in advance. In the fifth section, the protection of the system-wide secret private key using proactive secret sharing is presented using proactive secret sharing. The traffic created by these mechanisms is analysed for both threshold schemes that were identified to be suitable for the token-based accounting scheme. The sixth section analyses and compares alternative solutions for distributing the updates of the key shares among the trusted peers. As the best strategy, an improved version of the limited update and self initialisation method was identified. The final section of this chapter discusses failure recovery during transactions. Strategies are presented for resolution of situations during a transaction where the service delivery is interrupted or cancelled, as well as situations where there are disagreements about the transaction status.

Chapter 5 "System Evaluation" evaluates the token-based accounting scheme. After presenting the evaluation criteria and methodology, values for the parameters determining the token-based accounting schemes trustworthiness are determined using analytical considerations. Then the message complexity of the token-based accounting protocols is analysed, which are used to validate the simulation model. Section 4 presents the simulation model and analyses in detail the simulation results of token-based

accounting scheme. For each parameter influencing the created traffic (quorum size, account holder set size, churn, system size) four different experiments are performed. Based on the results, the influence of the parameters on the created traffic is derived. For the base scenario variable traffic per transaction is on average, 293.58 kBytes. This traffic includes payment traffic during a transaction as well as a fraction of token aggregation traffic. Further, this traffic is distributed among the peers participating in the transaction and token aggregation process. Constant traffic stems from maintenance of the account holder sets and is on average per peer 22.77 kBytes per minute. Assuming an upload bandwidth of 128 kBit per second in 97.85% of time peers have a load less than 1% of their upload capacity and in 99.35% of time peers have a load less than 10% of their upload capacity. Trusted peers have to bear sometimes (0.65% of the time) a higher load of approximately 14 seconds queue length. An analytical comparison with Karma, another accounting mechanism for p2p system (which is the most similar accounting mechanism to the token-based accounting scheme) shows that the token-based accounting scheme is distinctly more efficient. With a similar level of trustworthiness, Karma creates approximately double the amount of traffic per transaction, and approximately three times the amount of static maintenance traffic than the token-based accounting scheme. This proves the efficiency of the design of the token-based accounting scheme. Finally, the simulation model and the simulation results are presented for updating the key shares in a complete trusted peer system. Updating a complete system of trusted peers is fast and creates a low amount of traffic. Updating a system of 2000 trusted peers with a threshold of $t = 17$ requires approximately 68 seconds and creates on average 49.1 kBytes per peer in total. The highest traffic load per peer is 240.13 kBytes.

Chapter 6 "Deployment Issues" closes the research of the token-based accounting scheme by discussing deployment issues. First, the minimal requirements for deployment were analysed. Then alternatives for an deployment in commercial applications was presented. Three different charging alternatives, using tokens as receipts, using tokens as micropayment, and using tokens as bill of exchanged, are presented and analysed in terms of generated traffic depending on the required trustworthiness. The analysis shows that bills of exchange is the most efficient charging alternative. Last, the deployment and effects of using symmetric and asymmetric incentives in community applications like file sharing was analysed based on simulations. Using simulations, it is shown that in the presence of altruistic peers symmetric incentives lead to a distinctively lower system performance than asymmetric incentives. In order to keep the number of available tokens in the system stable, slightly asymmetric incentives are required. The degree of asymmetry depends on the ratio of altruistic peers in the system. The best system performance is achieved with stronger asymmetry, however this leads to inflation, which can lead to a malfunction of the incentive system.

7.2 Conclusion

Since the emergence of p2p systems is has been perceived that due to their decentralised and autonomous nature, trustworthy and efficient accounting is hard to achieve (if not impossible). In fact, in the early stages of the research presented in this thesis, many of these crucial challenges were identified and addressed.

Due to the novel ***token-based accounting scheme,*** it is demonstrated that these challenges can be solved and ***fully decentralised trusted accounting with intrinsic cooperation control*** is feasible and operational. This was achieved by

- *Designing and developing framework, system architecture, and protocols of the fully distributed, trusted token-based accounting scheme.*
  In order to collect accounting information, tokens are used as transaction receipts. The token-based accounting scheme consists of four building blocks, where Token Structure, Transaction Protocols, Token Aggregation, and Detection of Double Spending are the architecture's core components.

- *Designing a fully decentralised mechanism that allows automatic cooperation control within the token-based accounting scheme.*
  The possession of tokens gives the user permission to draw services from the p2p system. Within transactions, peers "spend" tokens by handing them over to the service provider. Peers swap foreign tokens for new own tokens in the token aggregation process. The *aggregation function* used to determine the number of new tokens adds another degree of freedom and can be used to express rules and policies in the p2p system.

- *Designing fully distributed mechanisms that achieve the trustworthiness of the token-based accounting without having to rely on a single trusted entity in the system.*
  Tokens are issued to peers in a fully distributed, trustworthy process using a quorum that establishes a *distributed basis of trust in p2p systems*. This distributed basis of trust is built by applying threshold cryptography and proactive secret sharing to p2p systems and combining it with a *novel mechanisms that ensure random quorum peer selection*. The efficient application of proactive secret sharing techniques to p2p systems is demonstrated using simulations.
  In order to prevent double spending, aggregation accounts are introduced that are replicated and stored at third party peers – the account holder set. Aggregation accounts are protected from cheating and collusion by *concealing their location using a novel routing mechanism* for structured p2p overlays. Maintenance protocols prevent loss of data and ensure consistency of aggregation accounts. The protocols have been proved to be efficient and robust under several scenarios of churn.
  Finally, a *novel transaction process* was developed *that removes the benefits of defrauding the transaction partner*. This ensures a trustworthy transaction process without using a third trusted party.

- Showing the feasibility of the scheme by demonstrating its *viability, trustworthiness, scalability, and efficiency* using analytical assessments as well as simulations. *The results show the applicability of the token-based accounting scheme in a wide variety of scenarios.* More specifically:

  - The required protocols are designed in detail, implemented in the peer-to-peer simulator "PeerfactSim.KOM", and the token-based accounting scheme is evaluated with it.

  - The trustworthiness of the token-based accounting scheme is demonstrated. The quorum size determines the scheme's trust level and is determined using a stochastic model. The required account holder set size was determined for various churn scenarios using simulations.

  - The token-based accounting scheme's efficiency is demonstrated by evaluating the traffic generated using simulations.
    The generated traffic is low compared to service traffic. Variable traffic per transaction increases linearly with the quorum size. Fixed maintenance traffic increases with a weak quadratic factor with an increase account holder set size.
    The system key maintenance process, using proactive secret sharing is fast and inexpensive traffic-wise. Therefore, it can be performed on daily basis, or even more often.
    An analytical comparison with the most relevant related work shows that the token-based accounting scheme is significantly more efficient, because it creates distinctively less traffic at approximately the same trust level.

  - The scalability of the scheme is evaluated by simulating different models of churn and systems sizes. Churn has only a weak influence on the generated fixed maintenance traffic. System size does not affect individual peer traffic. Therefore, there is no limit to scalability.

- *Discussing the applicability of the token-based accounting scheme in two scenarios*.
  It is shown, that the token-based scheme can be applied in commercial scenarios with several charging alternatives (such as receipts, micropayments, and bills of exchange). In community

applications with the aggregation function, asymmetric incentives can be given to peers, which influences their collaborative behaviour.

In the context of the project Market Management of Peer-to-Peer Services (MMAPPS) (MMA04) funded by the European Union and in the context of the project "Preis- und Erlösmodelle für das Internet – Umsetzung und Marktchancen" (PREMIUM) (Pro03) funded by the German Federal Ministry for Education and Research (BMBF), a prototype implementing large parts of the token-based accounting scheme was built and presented for example at CeBit 2006. A patent of the token-based accounting scheme has been applied for and is in the process of being granted (LDM04).

7.3 Outlook

The contributions made in this dissertation enable the implementation of new application scenarios using p2p systems, since the missing accounting functionality can now be feasibly provided. Furthermore, the novel mechanisms presented allow building more trustworthy p2p systems, which distinctly lowers the barrier for p2p systems to new application areas and towards seamless multimedia communications. This reveals new avenues of research, not only in the field of communication networks. In the following some of them are presented.

Like many other mechanisms for p2p systems presented in the literature, the token-based accounting scheme relies on a reputation mechanism. Reputation mechanisms for p2p systems have been researched for some years; however, no solution has been developed that would be broadly accepted. Furthermore, most reputation mechanisms for p2p systems are designed as a means of cooperation control. This is related to open research questions, if a reputation mechanism is required by the token-based accounting scheme, it has different requirements since the data it administers has a different context. Such a reputation mechanism has to manage information about violation of rules and needs to evaluate this information. Designing such a reputation mechanism is an interesting and a highly relevant and research topic.

The Trusted Platform Module (TPM) is thought to resolve many security concerns in information technology and especially in computer networking. Assuming the availability of the TPM at some peers or at all peers of a p2p system might change the premises for the token-based accounting scheme fundamentally. However, the autonomy of peers prevails; therefore, with TPM security and trustworthiness issues will change, but will not be finally solved. Still, taking the existence of TPM into account when designing mechanisms for trusted p2p systems has the potential to significantly enhance their efficiency.

The strong growth of p2p traffic is a challenge to ISPs (Hec04). This results in increased load on their network and increased traffic across domain borders. Thus, p2p traffic is a considerable cost factor within ISPs' operational costs. It is a new and interesting research field how p2p traffic can be controlled and guided. A solution to this challenge can be Economic Traffic Management, where ISPs give incentives to users to act "ISP friendly". An example for such an approach is the SmoothIT project (Smo08). The token-based accounting scheme could be used for Economic Traffic Management by rewarding peers with additional tokens for ISP-friendly behaviour. Thus, integrating the token-based accounting scheme in the economic concepts of Economic Traffic Management could be a highly valuable contribution to this field.

Using tokens in applications as means for incentives or as virtual currency, an economy of tokens can emerge. Accordingly, there are economic aspects to be considered that are similar to the economic challenges in the area of quantity theory of money. These concepts assume a central bank that controls the volume of money available. Hence, if such economic mechanisms are be applied in an application scenario there arise new technical challenges, e.g., how to determine the number of tokens available in the system. Furthermore, it is an interesting question, whether the economic theory created for countries and societies can be applied virtual communities. Thus, the combination of the token-based accounting

scheme with virtual communities puts existing economic theory into a new context and raises interesting new research challenges in the area of information management and economics.

Bibliography

[ABF+99] H. Appel, I. Biehl, A. Fuhrmann, M. Ruppert, T. Takagi, A. Takura, and C. Valentin: *Ein sicherer, robuster Zeitstempeldienst auf der Basis verteilter RSA-Signaturen*. Technical Report TR-21/99, Technische Universität Darmstadt, 1999.

[ABOS03] A. Agrawal, D. Brown, A. Ojha, and S. Savage: *Bucking Free-Riders: Distributed Accounting and Settlement in Peer-to-Peer Networks*. Technical Report UCSD Tech Report CS2003-0751, University of California, San Diego, 2003.

[ACM04] P. Antoniadis, C. Courcoubetis, and R. Mason: *Comparing Economic Incentives in Peer-to-Peer Networks*. Computer Networks, 46(1):133–146, 2004.

[ACS05] P. Antoniadis, C. Courcoubetis, and B. Strulo: *Incentives for Content Availability in Memory-less Peer-to-Peer File Sharing Systems*. ACM SIGecom Exchanges, 5(4):11–20, 2005.

[AH00] E. Adar and B. A. Huberman: *Free Riding on Gnutella*. First Monday, 5(10), 2000.

[AH02] K. Aberer and M. Hauswirth: *An Overview on Peer-to-Peer Information Systems*. In *Workshop on Distributed Data and Structures (WDAS-2002)*, 2002.

[BAM00] I. N. Bronstein, K. A.Semendjajew, and G. Musiol: *Taschenbuch der Mathematik*. Verlag Harri Deutsch, 2000.

[Bar01] D. Barkai: *Peer-to-Peer Computing. Technologies for Sharing and Collaboration on the Net.* Intel Press, 2001.

[BB03] A. Barmouta and R. Buyya: *GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration*. In *17th Annual International Parallel & Distributed Processing Symposium (IPDPS 2003) Workshop on Internet Computing and E-Commerce*, pages 22–26, 2003.

[BDH+03] B. Briscoe, V. Darlagiannis, O. Heckmann, H. Oliver, V. A. Siris, D. Songhurst, and B. Stiller: *A Market Managed Multi-Service Internet (M3I)*. Computer Communications, 26(4):404–414, 2003.

[BEH+01] R. Brussee, H. Eertink, W. Huijsen, B. Hulsebosch, M. Rougoor, W. Teeuw, M. Wibbels, and H. Zandbelt: *Content Distribution Networks - State of the Art*. Technical Report TI/RS/2001/027, Telematica Instituut, 2001.

[BF01] D. Boneh and M. Franklin: *Efficient Generation of Shared RSA keys*. Journal of the ACM (JACM), 48(4):702–722, 2001.

[BGG94] J. Bleymüller, G. Gehlert, and H. Gülicher: *Statistik für Wirtschaftswissenschaftler*. Verlag Vahlen, 1994.

[BHS02] F. Belanger, J. Hiller, and W. Smith: *Trustworthiness in Electronic Commerce: The Role of Privacy, Security, and Site Attributes*. The Journal of Strategic Information Systems, 11(3-4):245–270, 2002.

[BKLS02] P. Barreto, H. Y. Kim, B. Lynn, and M. Scott: *Efficient Algorithms for Pairing-Based Cryptosystems*. In *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 354–368, 2002.

[BKM08a] D. Bradler, J. Kangasharju, and M. Mühlhäuser: *Evaluation of Peer-to-Peer Overlays for First Response*. In *Mobile Peer-to-Peer Computing MP2P'08, in conjunction with the 6th IEEE International Conference on Pervasive Computing and Communications (PerCom'08)*, 2008.

[BKM08b] D. Bradler, J. Kangasharju, and M. Mühlhäuser: *Systematic First Response Use Case Evaluation*. In *Preceedings on 6th Workshop on Mobile and Distributed Approaches in Emergency Scenarios*, 2008.

[Bla79] G. Blakely: *Safeguarding Cryptographic Keys*. American Federation of Information Processing Societies Conference Proceedings, 48:313–317, 1979.

[BLS01] D. Boneh, B. Lynn, and H. Shacham: *Short Signatures from the Weil Pairing*. In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, pages 514–532, 2001.

[BLS04] D. Boneh, B. Lynn, and H. Shacham: *Short Signatures from the Weil Pairing*. Journal of Cryptology, 17(4):297–319, 2004.

[BMR+05] A. Brampton, A. MacQuire, I. A. Rai, N. J. P. Race, and L. Mathy: *Stealth Distributed Hash Table: Unleashing the Real Potential of Peer-to-Peer*. In *Proceeding of the ACM Conference on Emerging Network Experiments and Technology (CoNEXT) (Student Workshop Session)*, pages 230–231, 2005.

[Bol03] A. Boldyreva: *Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme*. In *PKC '03: Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *LNCS*, pages 31–46, 2003.

[Bon98] D. Boneh: *The Decision Diffie-Hellman Problem*. In *Proceedings of the Third Algorithmic Number Theory Symposium*, volume 1423 of *LNCS*, pages 48–63, 1998.

[BQ04] F. E. Bustamante and Y. Qiao: *Friendships that last: Peer Lifespan and its Role in P2P Protocols*. In *Proceedings of the 8th International Workshop on Web Content Caching and Distribution*, pages 233–246, 2004.

[BSPM08] D. Bradler, I. Schweizer, K. Panitzek, and M. Mühlhäuser: *First Response Communication Sandbox*. In *Proceedings of 11th Communications and Networking Simulation Symposium (CNS 2008)*, pages 115–122, 2008.

[Buc03] J. Buchmann: *Einführung in die Kryptographie*. Springer Verlag, 2003.

[Car08] Carnegie Mellon University, Software Engineering Institute: *Software Technology Roadmap, Keyword Index*. http://www.sei.cmu.edu/str/indexes/glossary/, 2008.

[CDK02] G. Coulouris, J. Dollimore, and T. Kindberg: *Verteilte Systeme - Konzepte und Design*. Pearson Studium, 3rd edition, 2002.

[CFN90] D. Chaum, A. Fiat, and M. Naor: *Untraceable Electronic Cash*. In *c*, volume 403 of *LNCS*, pages 319–327, 1990.

[Cha81] D. Chaum: *Untraceable Electronic Mail, Retrun Addresses, and Digital Pseudonyms*. Communications of the ACM, 4(2):84–90, 1981.

[Cha88] D. Chaum: *The Dining Cryptographers Problem: Unconditional Sender and Receipient Untraceability*. Journal of Cryptography, 1:65–75, 1988.

[Cho07] S. S. M. Chow: *Running on Karma – P2P Reputation and Currency Systems*. In *Cryptology and Network Security*, volume 4856 of *LNCS*, pages 146–158, 2007.

[Cia05] G. Ciaccio: *Evaluating Sender and Recipient Anonymity in a Structured Overlay*. Technical Report DISI-TR-05-13, Università di Genova, 2005.

[CKLS02] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl: *Asynchronous Verifiable Secret Sharing and Proactive Cryptosystems*. In *Proceedings of 9th ACM Conference on Computer and Communications Security (CCS)*, pages 88–97, 2002.

[CL99] M. Castro and B. Liskov: *Practical Byzantine Fault Tolerance*. In *Third Symposium on Operating Systems Design and Implementation (OSDI)*, pages 398–461, 1999.

[Cli08] Clip2, The Gnutella Developer Forum (GDF): *Gnutella Protocol Development*. http://rfc-gnutella.sourceforge.net/developer/stable/index.html, 2008.

[CLM03] M. Czyrnek, M. Luboński, and C. Mazurek: *Authentication, Authorisation and Accounting in Distributed Multimedia Content Delivery System*. In *Proceedings of TERENA Networking Conference*, 2003.

[Coh03] B. Cohen: *Incentives Build Robustness in BitTorrent*. In *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[CW03] C. Courcoubetis and R. Weber: *Pricing Communication Networks: Economics, Technology and Modelling*. John Wiley & Sons Ltd, 2003.

[DA06] Z. Despotovic and K. Aberer: *P2P Reputation Management: Probabilistic Estimation vs. Social Networks*. Computer Networks, 50(4):485–500, 2006.

[Dar05] V. Darlagiannis: *Overlay Network Mechanisms for Peer-to-Peer Systems*. PhD thesis, Technische Universität Darmstadt, 2005.

[DF89] Y. Desmedt and Y. Frankel: *Threshold Cryptosystems*. In *CRYPTO '89: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, volume 435 of *LNCS*, pages 307–315, 1989.

[DFM01] R. Dingledine, M. J. Freedman, and D. Molnar: *Peer-To-Peer: Harnessing the Power of Disruptive Technologies*, chapter Accountability, pages 217 – 340. O'Reilly & Associates, 1st edition, 2001.

[DGGZ03] D. Dutta, A. Goel, R. Govindan, and H. Zhang: *The Design of A Distributed Rating Scheme for Peer-to-peer Systems*. In *Proceedings of the Workshop on the Economics of Peer-to-Peer Systems*, 2003.

[Dou02] J. R. Douceur: *The Sybil Attack*. In *Proceedings of First International Workshop on Peer-to-Peer Systems*, volume 2429 of *LNCS*, 2002.

[DW04] B. T. David Wagner: *CS276 Cryptography*. Technical report, University of California Berkeley, 2004.

[Eck04] C. Eckert: *It-Sicherheit. Konzepte - Verfahren - Protokolle*. Oldenburg Verlag, 3rd edition, 2004.

[eMu04a] *eMule Project*. http://emule-project.net, 2004.

[eMu04b] eMule Project: *eMule's Credit System*. http://www.emule-project.net/home/perl/help.cgi?l=2&rm=show_topic&topic_id=69, 2004.

[FCC+03] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat: *SHARP: An Architecture for Secure Resource Peering*. In *SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 133–148, 2003.

[Fel87] P. Feldman: *A Practical Scheme for Non-Interactive Verifiable Secret Sharing*. Proceedings of the 28th IEEE Symposium of the Foundation of Computer Science, pages 427–437, 1987.

[FGMY97a] Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung: *Optimal-Resilience Proactive Public-Key Cryptosystems*. In *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, page 384, 1997.

[FGMY97b] Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung: *Proactive RSA*. In *CRYPTO '97: Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, volume 1294, pages 440–454, 1997.

[FLP85] M. J. Fischer, N. A. Lynch, and M. S. Paterson: *Impossibility of Distributed Consensus with one Faulty Process*. J. ACM, 32(2):374–382, 1985.

[FMY98] Y. Frankel, P. D. MacKenzie, and M. Yung: *Robust Efficient Distributed RSA-Key Generation*. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 663–672, 1998.

[FMY99] Y. Frankel, P. D. MacKenzie, and M. Yung: *Adaptively-Secure Optimal-Resilience Proactive RSA*. In *ASIACRYPT*, pages 180–194, 1999.

[FSCM02] M. J. Freedman, E. Sit, J. Cates, and R. Morris: *Introducing Tarzan: A Peer-to-Peer Anonymizing Network Layer*. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, pages 121 – 129, 2002.

[Gal02] C. Gale: *User Motivations, Trust, Anonymity and Abuse-Prevention in P2P-Systems Services. Deliverable 3 of the MMAPPS Project*. Technical report, The MMAPPS Consortium, 2002.

[Gam85] T. E. Gamal: *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*. IEEE Transactions on Information Theory, 31:469 – 472, 1985.

[GH05] F. D. Garcia and J.-H. Hoepman: *Off-Line Karma: A Decentralized Currency for Peer-to-peer and Grid Applications*. In *Applied Cryptography and Network Security*, volume 3531 of *LNCS*, pages 364–377, 2005.

[GJKR99] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin: *Secure Distributed Key Generation for Discrete-Log Based Cryptosystems*. In J. Stern (editor): *Advances in Cryptology - EUROCRYPT '99*, volume 1592 of *LNCS*, pages 295–310, 1999.

[GJKR01] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin: *Robust Threshold DSS signatures*. Information and Computation, 164(1):54–84, 2001.

[GJKR03] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin: *Secure Applications of Pedersen's Distributed Key Generation Protocol*. In *Topics in Cryptology – CT-RSA 2003*, volume 2612 of *LNCS*, pages 373–390, 2003.

[GJKR07] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin: *Secure Distributed Key Generation for Discrete-Log Based Cryptosystems.* Journal of Cryptology, 20(1):51–83, 2007.

[GR06] M. Göhner and C.-P. Rückemann: *Konzeption eines Grid-Accounting-Systems*. Technical report, D-Grid, 2006.

[GWG+07] M. Göhner, M. Waldburger, F. Gubler, G. D. Rodosek, and B. Stiller: *An Accounting Model for Dynamic Virtual Organizations*. In *Proceedings of the 17th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007)*, pages 241–248, 2007.

[Has05] G. Hasslinger: *Peer-to-Peer Systems and Applications*, volume 3485 of *LNCS*, chapter ISP Platforms under a Heavy Peer-to-Peer Workload, pages 369–382. Springer, 2005.

[Hau05] D. K. Hausheer: *PeerMart: Secure Decentralized Pricing and Accounting in Peer-to-Peer Networks*. PhD thesis, ETH Zürich, 2005.

[HD05] M. Hauswirth and S. Dustdar: *Peer-to-Peer: Grundlagen und Architektur*. Datenbank Spektrum, 13:5–13, 2005.

[Hec04] O. Heckmann: *A System-Oriented Approach to Efficiency and Quality of Service for Internet Service Providers*. PhD thesis, Technische Universtät Darmstadt, Germany, 2004.

[HGS05] D. Hausheer, J. Gerke, and B. Stiller: *A Generic and Modular Accounting and Charging System for Peer-to-Peer Applications*. In *14. Fachtagung Kommunikation in Verteilten Systemen 2005 (KiVS 05)*, 2005.

[HJJK97] A. Herzberg, M. Jakobsson, S. Jarecki, and H. Krawczyk: *Proactive Public Key and Signature Systems*. In *Proceedings of ACM Conference on Computer and Communications Security*, pages 100–110, 1997.

[HJKY95] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung: *Proactive Secret Sharing Or: How to Cope With Perpetual Leakage*. In *Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '95)*, volume 963 of *LNCS*, pages 339–352, 1995.

[HLD+05] O. Heckmann, N. Liebau, V. Darlagiannis, A. Bock, A. Mauthe, and R. Steinmetz: *From Integrated Publication and Information Systems to Information and Knowledge Environments: Essays Dedicated to Erich J. Neuhold on the Occasion of His 65th Birthday*, volume 3379 of *Lecture Notes in Computer Science*, chapter A Peer-to-Peer Content Distribution Network, pages 69–78. Springer-Verlag GmbH, 2005.

[Hol04] M. Hollick: *Dependable Routing for Cellular and Ad hoc Networks*. PhD thesis, Technische Universität Darmstadt, 2004.

[HS05a] D. Hausheer and B. Stiller: *Decentralized and Secure Accounting for Peer-to-Peer Applications*. In *2005 IFIP Networking Conference*, volume 3462 of *LNCS*, pages 40–52, 2005.

[HS05b] D. Hausheer and B. Stiller: *Decentralized Auction-Based Pricing with PeerMart*. In *Proceedings of the 9th IFIP/IEEE International Symposium on Integrated Network Management (IM 2005)*, 2005.

[HW02] S. Hazel and B. Wiley: *Achord: A Variant of the Chord Lookup Service for use in Censorship Resistant Peer-to-Peer Publishing Systems*. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, 2002.

[Jai91] R. Jain: *Art of Computer Systems Performance Analysis Techniques For Experimental Design Measurements Simulation And Modeling*. Wiley Computer Publishing, John Wiley & Sons, Inc., 1991.

[JJ99] M. Jakobsson and A. Juels: *Communications and Multimedia Security*, chapter Proofs of Work and Bread Pudding Protocols, pages 258–272. Kluwer Academic Publishers, 1999.

[JS05] S. Jarecki and N. Saxena: *Further Simplifications in Proactive RSA Signatures*. In *Proceedings of Theory of Cryptography Conference'05*, pages 510–528, 2005.

[JSY04] S. Jarecki, N. Saxena, and J. H. Yi: *An Attack on the Proactive RSA Signature Scheme in the URSA Ad hoc Network Access Control Protocol*. In *SASN '04: Proceedings of the 2nd ACM workshop on Security of Ad Hoc and Sensor Networks*, pages 1–9, 2004.

[JXT04] *Crypto-ID Project*. http://crypto-id.jxta.org, 2004.

[Kar00] M. Karsten: *QoS Signalling and Charging in a Multi-service Internet using RSVP*. PhD thesis, Technische Universität Darmstadt, 2000.

[KDG03] D. Kempe, A. Dobra, and J. Gehrke: *Gossip-Based Computation of Aggregate Information*. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 482–491, 2003.

[KIKOU08] M. Karakaya, İbrahim Körpeoğlu, and Özgür Ulusoy: *Counteracting Free Riding in Peer-to-Peer Networks*. Computer Networks: The International Journal of Computer and Telecommunications Networking, 52(3):675–694, 2008.

[KKM+07] A. Kovacevic, S. Kaune, P. Mukherjee, N. Liebau, and R. Steinmetz: *Benchmarking Platform for Peer-to-Peer Systems*. it - Information Technology (Methods and Applications of Informatics and Information Technology), 49(5):312–319, 2007.

[Kne07] P. Knežević: *High Data Availability and Consistency for Distributed Hash Tables Deployed in Dynamic Peer-to-Peer Communities*. PhD thesis, Technische Universität Darmstadt, 2007.

[KRR02] J. Kangasharju, J. Roberts, and K. W. Ross: *Object Replication Strategies in Content Distribution Networks*. Computer Communications, 25(3):367–383, 2002.

[KRS03] M. Karsten, P. Reichl, and B. Stiller: *Internet Pricing and Charging*. Computer Communications, 26(13):1433, 2003.

[KSGM03] S. Kamvar, M. Schlosser, and H. Garcia-Molina: *EigenRep: Reputation Management in P2P Networks*. In *Proceedings of the 12th International World Wide Web Conference*, 2003.

[KSTT04] R. Krishnan, M. D. Smith, Z. Tang, and R. Telang: *The Impact of Free-Riding on Peer-to-Peer Networks*. In *HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 7*, page 70199.3, 2004.

[KZL+01] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang: *Providing Robust and Ubiquitous Security Support for Mobile Ad-hoc Networks*. In *Proceedings of IEEE 9th International Conference on Network Protocols (ICNP'01)*, pages 251–260, 2001.

[Lam98] L. Lamport: *The Part-Time Parliament*. ACM Transactions on Computer Systems, 16(2):133–169, 1998.

[Lam01] L. Lamport: *Paxos Made Simple*. ACM SIGACT News, 32(4):18–25, 2001.

[LDM04] N. Liebau, V. Darlagiannis, and A. Mauthe: *Ein Dezentrales, Token-Basiertes Accountingsystem für Verteilte, Autonome Systeme*. European Patent Application 04 101 386.3 - 1 477 882, 2004.

[LDMS05] N. Liebau, V. Darlagiannis, A. Mauthe, and R. Steinmetz: *Token-based Accounting for P2P-Systems*. In *Proceeding of Kommunikation in Verteilten Systemen KiVS 2005*, pages 16–28, 2005. (Received Best Paper Award).

[LKZ+04] H. Luo, J. Kong, P. Zerfos, S. Lu, and L. Zhang: *URSA: Ubiquitous and Robust Access Control for Mobile Ad Hoc Networks*. IEEE/ACM Transactions on Networking, 12(6):1049–1063, 2004.

[LL00] H. Luo and S. Lu: *Ubiquitous and Robust Authentication Services for Ad Hoc Wireless Networks*. Technical Report UCLA-CSD-TR-200030, University of California, Los Angeles, 2000.

[LLZ04] S. Lin, M. Lian, and Z. Zhang: *A Practical Distributed Mutual Exclusion Protocol in Dynamic Peer-to-Peer Systems*. In *Proceedings of 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04)*, 2004.

[LPG+07] N. Liebau, K. Pussep, K. Graffi, S. Kaune, E. Jahn, A. Beyer, and R. Steinmetz: *The Impact of the P2P Paradigm on the New Media Industries*. In *Proceedings of Americas Conference on Information Systems*, 2007.

[LSP82] L. Lamport, R. Shostak, and M. Pease: *The Byzantine Generals Problem*. ACM Transactions on Programming Languages and Systems, 4(3):382–401, 1982.

[MAD02] J.-P. Martin, L. Alvisi, and M. Dahlin: *Small Byzantine Quorum Systems*. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 374–388, 2002.

[Mar05] J.-P. Martin: *Fast Byzantine Consensus*. In *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 402–411, 2005.

[Mer08] Merriam-Webster, Inc.: *Merriam-webster dictionary*. http://www.merriam-webster.com/dictionary/, 2008.

[Met04] MetaMachine: *eDonkey2000*. http://www.edonkey2000.com, 2004.

[MLTS08] P. Mukherjee, C. Leng, W. Terpstra, and A. Schürr: *Peer-to-Peer based Version Control*. In *Proceedings of the P2PNVE '08*, 2008. to appear.

[MM02] P. Maymounkov and D. Maziéres: *Kademlia: A Peer-to-Peer Information System Based on the XOR Metric*. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02)*, 2002.

[MMA04] MMAPPS: *Project "Market Management of Peer-to-Peer Services"; project homepage*. http://www.mmapps.info, 2004.

[mne05] mnet: *mnet Project Homepage*. http://mnetproject.org/, 2005.

[Moj00] MojoNation: *MojoNation Technology Overview*. http://surfvi.com/ ota/others-papers/MojoNation_TechnicalOverview.html, 2000. Last Viewed 07/2008.

[MOP+04] A. Mislove, G. Oberoi, A. Post, C. Reis, P. Druschel, and D. Wallach: *AP3: A cooperative, Decentralized Service Providing Anonymous Communication*. In *Proceedings of the 11th ACM SIGOPS European Workshop*, 2004.

[MOR01] S. Micali, K. Ohta, and L. Reyzin: *Accountable-Subgroup Multisignatures: Extended Abstract*. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 245–254, 2001.

[MR97] D. Malkhi and M. Reiter: *Byzantine Quorum Systems*. In *STOC '97: Proceedings of the 29th annual ACM Symposium on Theory of Computing*, pages 569–578, 1997.

[MS07] P. Mahlmann and C. Schindelhauer: *Peer-to-Peer Netzwerke – Algorithm und Methoden*. Springer, 2007.

[MT03] T. Moreton and A. Twigg: *Trading in Trust, Tokens, and Stamps*. In *Proceedings of the 1st Workshop on the Economics of Peer-to-Peer Systems*, 2003.

[Mul] Multimedia Communications Lab – Technische Universität Darmstadt: *PeerfactSim.KOM*. http://peerfact.kom.e-technik.tu-darmstadt.de/.

[MVHE04] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg: *Local-lag and Timewarp: Providing Consistency for Replicated Continuous Applications*. IEEE Transactions on Multimedia, 6(1):47 – 57, 2004.

[Nat94] National Institute of Standards and Technology: *Digital Signature Standard*. FIPS PUB 186, 1994.

[Nat99] National Security Telecommunications and Information Systems Security Committee: *National Information Systems Security (INFOSEC) Glossary*. Technical Report NSTISSI No. 4009, National Security Telecommunications and Information Systems Security Committee, 1999.

[Neu93] B. C. Neumann: *Proxy-Based Authorization and Accounting for Distributed Systems*. In *Proceedings of International Conference on Distributed Computing Systems*, pages 283–291, 1993.

[NN04] V. Nikov and S. Nikova: *On Proactive Secret Sharing Schemes*. In *Selected Areas in Cryptography*, pages 308–325, 2004.

[NNPV02a] V. Nikov, S. Nikova, B. Preneel, and J. Vandewalle: *Applying General Access Structure to Proactive Secret Sharing Schemes*. In *Proceedings of the 23rd Symposium on Information Theory in the Benelux*, pages 197–206, 2002.

[NNPV02b] V. Nikov, S. Nikova, B. Preneel, and J. Vandewalle: *On Distributed Key Distribution Centers and Unconditionally Secure Proactive Verifiable Secret Sharing Schemes Based on General Access Structure*. In *INDOCRYPT '02: Proceedings of the Third International Conference on Cryptology*, pages 422–436, 2002.

[NNS+04] T.-W. Ngan, A. Nandi, A. Singh, D. S. Wallach, and P. Druschel: *On Designing Incentives-Compatible Peer-to-Peer Systems*. In *Proceedings of 2nd Bertinoro Workshop on Future Directions in Distributed Computing (FuDiCo II: S.O.S.)*, 2004.

[NT04] N. Ntarmos and P. Triantafillou: *SeAl: Managing Accesses and Data in Peer-to-Peer Sharing Networks*. In *Proceedings of the 4th IEEE International Conference on Peer-to-Peer Computing*, 2004.

[Oka88] T. Okamoto: *A Digital Multisignature Scheme using Bijective Public-Key Cryptosystems*. ACM Transactions on Computer Systems (TOCS), 6(4):432–441, 1988.

[On04] G. On: *Quality of Availability for Widely Distributed and Replicated Content Stores*. PhD thesis, Technische Universität Darmstadt, 2004.

[Ped91a] T. P. Pedersen: *A Threshold Cryptosystem Without a Trusted Party*. In D. Davies (editor): *Advances in Cryptology - EUROCRYPT '91*, volume 547 of *LNCS*, pages 522–526, 1991.

[Ped91b] T. Pedersen: *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing*. CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology, pages 129–140, 1991.

[Pei01] M. Peirce: *Payment mechanisms designed for the Internet*. http://ntrg.cs.tcd.ie/mepeirce/Project/oninternet.html, 2001.

[Pro03] Project Premium: *Preis- und Erlösmodelle für das Internet – Umsetzung und Markchancen; Project Homepage*. http://www.internetoekonomie.uni-frankfurt.de/, 2003.

[PSL80] M. Pease, R. Shostak, and L. Lamport: *Reaching Agreement in the Presence of Faults*. Journal of the ACM, 27(2):228–234, 1980.

[PSW04] J. Pieprzyk, R. Steinfeld, and H. Wang: *Lattice-based Threshold-Changeability for Standard Shamir Secret-Sharing Schemes*. In *Proceedings of Asiacrypt 2004*, volume 3329 of *LNCS*, pages 170–186, 2004.

[Rab98] T. Rabin: *A Simplified Approach to Threshold and Proactive RSA*. In *CRYPTO '98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, volume 1462 of *LNCS*, pages 89–104, 1998.

[RAD00a] *RFC 2865: Remote Authentication Dial In User Service (RADIUS)*. http://www.freeradius.org/rfc/rfc2865.html, 2000.

[RAD00b] *RFC 2866: RADIUS Accounting*. http://www.freeradius.org/rfc/rfc2866.html, 2000.

[RBR+04] M. Roussopoulos, M. Baker, D. S. H. Rosenthal, T. Giuli, P. Maniatis, and J. Mogul: *2 P2P or Not 2 P2P?* In *Proceedings of the Third International Workshop on Peer-to-Peer Systems (IPTPS '04)*, volume 3279 of *LNCS*, pages 33–43, 2004.

[RD01] A. Rowstron and P. Druschel: *Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems*. In *IFIP/ACM Middleware*, pages 329–350, 2001.

[Röd02] U. Rödig: *Firewall-Architekturen für Multimedia-Applikationen*. PhD thesis, Technische Universität Darmstadt, Germany, 2002.

[RFS07] P. Racz, V. Forster, and B. Stiller: *Design and Implementation of an Integrated Accounting Architecture for Distributed UMTS and WLAN Networks*. International Journal of Wireless and Mobile Computing, 2(4):275–287, 2007.

[RM06] J. Risson and T. Moors: *Survey of Research towards Robust Peer-to-Peer Networks: Search Methods*. Computer Networks: The International Journal of Computer and Telecommunications Networking, 50(17):3485–3521, 2006.

[RS96] R. L. Rivest and A. Shamir: *PayWord and MicroMint: Two Simple Micropayment Schemes*. In *Security Protocols Workshop*, pages 69–87, 1996.

[RSA04] RSA Laboratories: *Techniques in Cryptography*, 2004.

[RSG98] M. G. Reed, P. F. Syverson, and D. M. Goldschlag: *Anonymous Connections and Onion Routing*. IEEE Journal on Selected Areas in Communication, 16(4):482–494, 1998.

[Sax06] N. Saxena: *Decentralized Security Services*. PhD thesis, University of California, Irvine, 2006.

[SBGP99] B. Stiller, T. Braun, M. Günter, and B. Plattner: *The CATI Project: Charging and Accounting Technology for the Internet*. In *Proceedings of 4th European Conference on Multimedia Applications, Services and Techniques - ECMAST'99*, volume 1629 of *LNCS*, pages 281–296, 1999.

[Sch91] C. P. Schnorr: *Efficient Identification and Signatures for Smart Cards*. In G. Brassard (editor): *Crypto '89: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, volume 435 of *LNCS*, pages 239–252, 1991.

[Sch96] B. Schneier: *Applied Cryptography*. John Wiley & Sons, 2nd edition, 1996.

[Sch98] B. Schoenmakers: *Basic Security of the ecashTM Payment System*. In B. Preneel and V. Rijmen (editors): *Course on Computer Security and Industrial Cryptography*, volume 1528 of *LNCS*, chapter State of the Art in Applied Cryptography, pages 338–352. Springer, 1998.

[Sch00] J. Schmitt: *Heterogeneous Network Quality of Service Systems*. PhD thesis, Technische Universität Darmstadt, 2000.

[Sch04] D. Schoder: *Suitability of P2P for Business Transactions*. In *Proceedings of the Peer-to-Peer Systems and Applications Daghstuhl Seminar*, 2004.

[SENB07a] M. Steiner, T. En-Najjary, and E. W. Biersack: *A Global View of KAD*. In *IMC '07: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, pages 117–122, 2007.

[SENB07b] M. Steiner, T. En-Najjary, and E. W. Biersack: *Analyzing Peer Behavior in KAD*. Research report RR-07-205, Institut Eurecom, Department of Corporate Communications, 2007.

[SF02] D. Schoder and K. Fischbach: *Peer-to-Peer: Ökonomische, technische und juristische Perspektiven*, chapter Peer-to-Peer. Anwendungsbereiche und Herausforderungen, pages 3–21. Springer Verlag, Heidelberg, 2002.

[SFPW98a] B. Stiller, G. Fankhauser, B. Plattner, and N. Weiler: *Charging and Accounting for Integrated Internet Services - State of the Art, Problems, and Trends*. In *The Internet Summit (INET 98)*, pages 21–24, 1998.

[SFPW98b] B. Stiller, G. Fankhauser, B. Plattner, and N. Weiler: *Pre-study on Customer Care, Accounting, Charging, Billing, and Pricing*. Technical report, TIK Laboratory, ETH Zurich, 1998.

[SGE+04] T. Sandholm, P. Gardfjäll, E. Elmroth, L. Johnsson, and O. Mulmo: *An OGSA-based accounting system for allocation enforcement across HPC centers*. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 279–288, 2004.

[SGF+01] B. Stiller, J. Gerke, P. Flury, P. Reichl, and Hasan: *Charging Distributed Services of a Computational Grid Architecture*. In *Proceedings of the First IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01)*, pages 596–601, 2001.

[Sha] Sharman Networks Limited: *Accounting in KaZaA*. http://www.kazaa.com/us/help/glossary/participation_ratio.htm.

[Sha79] A. Shamir: *How to Share a Secret*. Communications of the ACM, 22(11):612–613, 1979.

[Sha01] C. E. Shannon: *A Mathematical Theory of Communication*. SIGMOBILE Mob. Comput. Commun. Rev., 5(1):3–55, 2001.

[Sha04] Sharman Networks Limited: *KaZaA*. http://www.kazaa.com, 2004.

[Shi01] C. Shirky: *Peer to Peer: Harnessing the Power of Disruptive Technologies*, chapter Listening to Napster, pages 19–28. O'Reilly & Associates, 2001.

[SHL+06] R. Steinmetz, O. Heckmann, N. Liebau, A. Buchmann, C. Eckert, J. Kangasharju, M. Mühlhäuser, and A. Schürr: *Qualitätsmerkmale von Peer-to-Peer-Systemen*. Technical Report KOM-TR-2006-03, Multimedia Communications Lab, Technische Universität Darmstadt, 2006.

[Sho00] V. Shoup: *Practical Threshold Signatures*. In *Proceedings of Eurocrypt 2000*, 2000.

[SL04] A. Singh and L. Liu: *Agyaat: Providing Mutually Anonymous Services over Structured P2P Networks*. Technical Report GIT-CERCS-04-12, Georgia Institute of Technology, 2004.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan: *Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications*. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 149–160, 2001.

[SMLN+03] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan: *Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications*. IEEE/ACM Transactions on Networking, 11(1):17 – 32, 2003.

[Smo08] SmoothIT-Project: *SmoothIT - Simple Economic Management Approaches of Overlay Traffic on Heterogeneous Internet Topologies; Project Homepage*. http://www.ict-smoothit.eu/, 2008.

[SR05] D. Stutzbach and R. Rejaie: *Characterizing Churn in Peer-to-Peer Networks*. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 402–403, 2005.

[SRG+03] B. Stiller, P. Reichl, J. Gerke, Hasan, and P. Flury: *Charging and Accounting in High-speed Networks*. Future Generation Computer Systems, 19(1):101–109, 2003.

[ST03] A. Shamir and E. Tromer: *Factoring Large Number with the TWIRL Device.* In *Proceedings of Crypto 2003*, volume 2729 of *LNCS*, pages 1–26, 2003.

[Sta02] State Archives Department of the Minnesota Historical Society: *Trustworthy Information Systems Handbook*. Technical report, Minnesota Historical Society, 2002.

[STY03] N. Saxena, G. Tsudik, and J. H. Yi: *Admission Control in Peer-to-Peer: Design and Performance Evaluation*. In *SASN '03: Proceedings of the 1st ACM Workshop on Security of Ad Hoc and Sensor Networks*, pages 104–113, 2003.

[STY07] N. Saxena, G. Tsudik, and J. H. Yi: *Threshold Cryptography in P2P and MANETs: The Case of Access Control*. Computer Networks: The International Journal of Computer and Telecommunications Networking, 51(12):3632–3649, 2007.

[Sun04] Sun Microsystems: *Project JXTA*. http://www.jxta.org, 2004.

[SW99] D. R. Stinson and R. Wei: *Unconditionally Secure Proactive Secret Sharing Scheme with Combinatorial Structures*. In *Proceedings of the 6th Annual International Workshop on Selected Areas in Cryptography*, pages 200–214, 1999.

[SW04] R. Steinmetz and K. Wehrle: *Peer-to-Peer-Networking and -Computing*. Informatik Spektrum, 27(1):51–54, 2004.

[SW05a] R. Steinmetz and K. Wehrle: *Peer-to-Peer Systems and Applications*, volume 3485 of *LNCS*, chapter What is Peer-to-Peer About?, pages 9–16. Springer, 2005.

[SW05b] R. Steinmetz and K. Wehrle (editors): *Peer-to-Peer systems and Applications*. Springer-Verlag, 2005.

[Tan03] A. S. Tanenbaum: *Computer Networks*. Pearson Studium, 4th edition, 2003.

[TBLB06] B. Temkow, A.-M. Bosneag, X. Li, and M. Brockmeyer: *PaxonDHT: Achieving Consensus in Distributed Hash Tables*. In *SAINT '06: Proceedings of the International Symposium on Applications on Internet*, pages 236–244, 2006.

[THe06] *The free dictionary*. http://encyclopedia.thefreedictionary.com/, 2006.

[THMA02] W. Thigpen, T. J. Hacker, L. F. McGinnis, and B. D. Athey: *Distributed Accounting on the Grid*. In *Proceedings of the 6th Joint Conference on Information Sciences*, pages 1147–1150, 2002.

[TMOO05] R. Tso, Y. Miao, T. Okamoto, and E. Okamoto: *A Share-Correctable Protocol for the Shamir Threshold Scheme and Its Application to Participant Enrollment*. Information Processing Socienty of Japan Journal, 46(8):313–321, 2005.

[TOC+00] G. Tomlinson, H. Orman, M. Condry, J. Kempf, and D. Farber: *Extensible Proxy Services Framework*. Internet-Draft, Network Working Group, 2000.

[TPM04] K. Tamilman, V. Pai, and A. Mohr: *SWIFT: A System With Incentives For Trading*. In *Proceedings of Second Workshop of Economics in Peer-to-Peer Systems*, 2004.

[UC 08] UC Berkeley, Computer Science Division: *The OceanStore Project*. http://oceanstore.cs.berkeley.edu/, 2008.

[Uni08] University of California: *Seti@Home Home*. http://setiathome.berkeley.edu/, 2008.

[Var91] B. Varley: *Usage Accounting in Distributed Systems*. In *IEEE Colloquium on Network Management*, pages 7/1–7/8, 1991.

[VCS03] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer: *KARMA: A Secure Economic Framework for Peer-to-Peer Resource Sharing*. In *Proceedings of the Workshop on the Economics of Peer-to-Peer Systems*, 2003.

[Wal02] D. S. Wallach: *A Survey of Peer-to-Peer Security Issues*. In *International Symposium on Software Security*, pages 42–57, 2002.

[Web08] WebFinance, Inc.: "*payment*", *InvestorWords.com*. http://www.investorwords.com/3634/payment.html, 2008.

[Wik08a] Wikipedia: *KaZaA Lite-Varianten*. http://de.wikipedia.org/wiki/Kazaa-Lite-Varianten, 2008.

[Wik08b] Wikipedia: *Mnet*. http://en.wikipedia.org/wiki/Mnet, 2008.

[Wik08c] Wikipedia: *Napster*. http://de.wikipedia.org/wiki/Napster, 2008.

[WWW02] T. M. Wong, C. Wang, and J. M. Wing: *Verifiable Secret Redistribution for Archive Systems*. In *Proceedings of the First International IEEE Security in Storage Workshop (SISW 2002)*, pages 94–112, 2002.

[XL04] L. Xiong and L. Liu: *PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities*. IEEE Transactions on Knowledge and Data Engineering, 16(7):843–857, 2004.

[YGM02] B. Yang and H. Garcia-Molina: *Improving Search in Peer-to-Peer Networks*. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 5–14, 2002.

[YGM03] B. Yang and H. Garcia-Molina: *PPay: Micropayments for Peer-to-Peer Systems*. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, pages 300–310, 2003.

[YLLH04] J. Yu, M. Li, Y. Li, and F. Hong: *An Economy-Based Accounting System for Grid Computing Environments*. In *Workshop on Web Information Systems; Track1: Fragmentation Versus Integration - Perspective of the Web Information System Discipline (FIPWIS)*, volume 3307 of *LNCS*, pages 233–238, 2004.

[YS00] B. Yu and M. P. Singh: *A Social Mechanism of Reputation Management in Electronic Communities*. In *CIA '00: Proceedings of the 4th International Workshop on Cooperative Information Agents IV, The Future of Information Agents in Cyberspace*, pages 154–165, 2000.

[ZSvR02] L. Zhou, F. B. Schneider, and R. van Renesse: *COCA: A Secure Distributed On-line Certification Authority*. ACM Transactions on Computer Systems, 20(4):329–368, 2002.

All web pages cited in this work have been checked in June 2008. However, due to the dynamic nature of the World Wide Web, web pages can change.